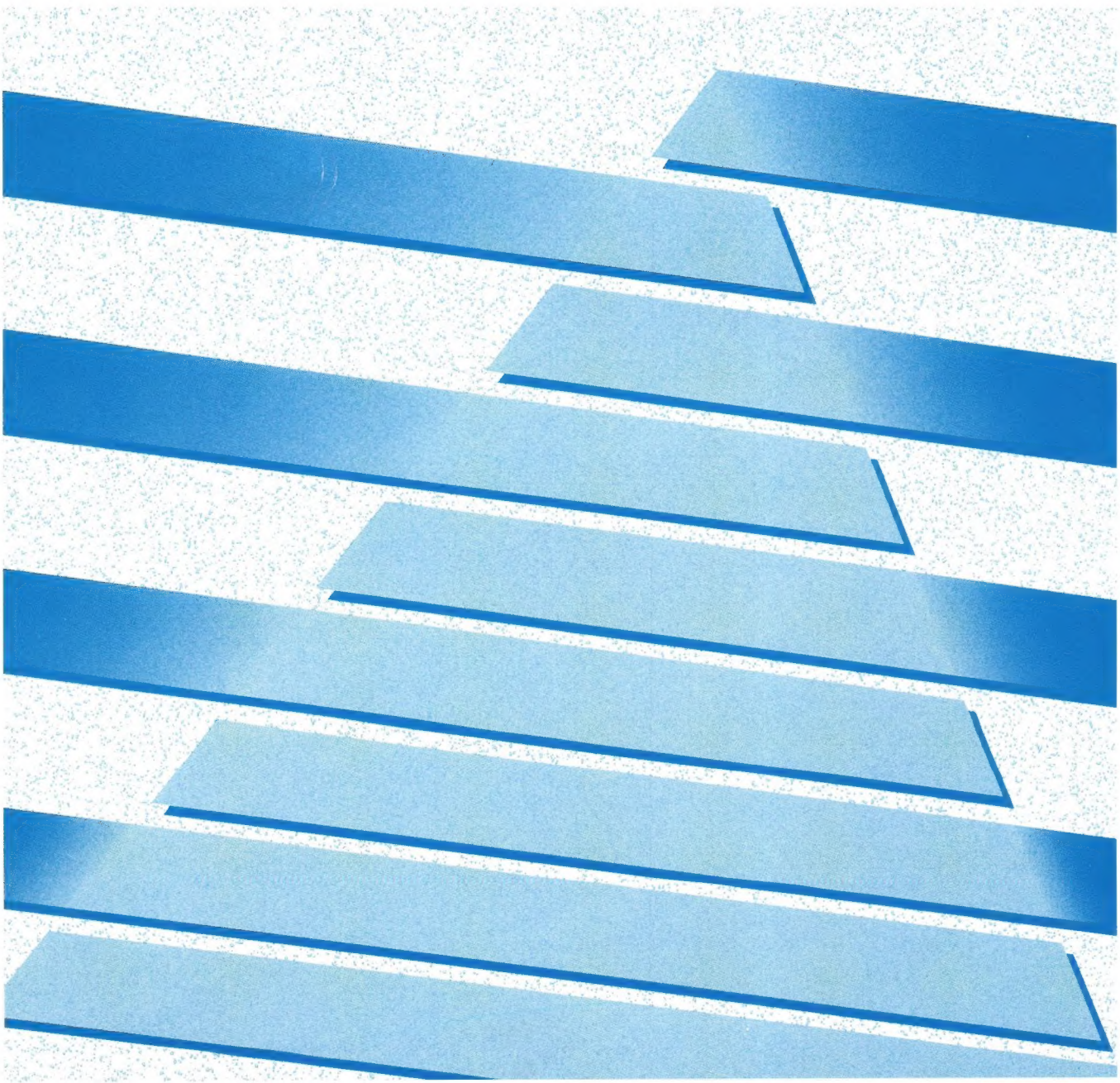
ALLEN-BRADLEY

ControlView™ Reporting

(Cat. No. 6190-REP)

User Manual



Important User Information

Because of the variety of uses for the products described in this publication, those responsible for the application and use of this control equipment must satisfy themselves that all necessary steps have been taken to assure that each application and use meets all performance and safety requirements, including any applicable laws, regulations, codes and standards.

The illustrations, charts, sample programs and layout examples shown in this guide are intended solely for purposes of example. Since there are many variables and requirements associated with any particular installation, the Allen-Bradley Company, Inc. does not assume responsibility or liability (to include intellectual property liability) for actual use based upon the examples shown in this publication.

Allen-Bradley Publication SGI-1.1, "Safety Guidelines for the Application, Installation and Maintenance of Solid State Control" (available from your local Allen-Bradley office) describes some important differences between solid-state equipment and electromechanical devices which should be taken into consideration when applying products such as those described in this publication.

Throughout this manual we use notes to make you aware of safety considerations:

ATTENTION: Identifies information about practices or circumstances that can lead to personal injury or death, property damage or economic loss.

Attentions help you:

- identify a hazard
- avoid the hazard
- recognize the consequences

Important: Identifies information that is especially important for successful application and understanding of the product.

Summary of Changes

Changes from Release 2.0 to 3.0

The following changes have been made to the Reporting option and the Reporting User Manual since release 2.0:

For Information on this new feature:	Refer to:	The feature appeared in:
Installation instructions for the Reporting option have been moved to the *ControlView Installation Manual.*	ControlView Installation Manual	software release 3.0
Generate Data Logger data subsets before a report is printed using the @rpt_generate keyword	Chapter 3	software release 3.0
Create @rpt_no_source data subsets to initialize memory variables	Chapter 2, Chapter 3	software release 3.0
Extract the time and date from certain fields or tags with four new functions: TAD_OF_MIN, TAD_OF_MAX, TAD_OF_FIRST_REC, and TAD_OF_LAST_REC	Chapter 3	software release 2.12
Specify time and date information for a report at run time (rather than defining these values inside the report's definition) with the /T parameter	Chapter 3, Appendix B	software release 2.12
Generate multiple versions of a report with the /C and /M parameters	Chapter 3, Appendix B	software release 2.12
Use the Report Completed message to find the name of the report output file or alternatively use the /S parameter to suppress the Report Completed message	Appendix B	software release 2.12
Specify a ControlView command that is run when the report has finished printing with the new Action field	Chapter 2	software release 2.12
Type place holders in the Action field so that the actual parameters can be specified at run time	Chapter 3	software release 2.12
Suppress printing of a table if the table's data subset is empty	Chapter 2	software release 2.12
Use database tags inside a table and/or data subsets	Chapter 1	software release 2.12
Define and use memory variables with the new SET_VAR function	Chapter 3	software release 2.12

Preface

How To Use This Manual

This manual describes the features and capabilities of the Reporting option, a component of the ControlView™ system. Reporting allows you to produce reports based on real time or historical data.

This manual supplements the information in the *ControlView Core User Manual.*

Conventions Used In This Manual

This manual follows the print conventions outlined in the *ControlView Core User Manual.*

Audience

Since Reporting is part of ControlView, you should be familiar with ControlView and have the *ControlView Core User Manual* available for reference. A complete list of related publications is contained in that manual.

Glossary

Data Subset—A temporary file, created when the report is generated, that contains data to be included in the report. A data subset may be a copy of an entire log file, although it will usually contain only selected data. Data subsets are only one source of data for a report, the others being ControlView's database and memory variables.

Field, in a template—A placeholder within the report's template that identifies what data to use at that location in the report and what format to present it in when the report is generated.

Field, in a log file—One element of data within a log file.

Icon—A white line displayed on the Report Editor's screen to depict the boundaries of certain reporting features. Icons are used to mark the beginning and end of a table and summary and to illustrate a page break.

Log File—ControlView data files (also referred to as "file sets") used to store historical data. There are three types of log files: the alarm log file, the activity log file and log files created by various Data Logger models.

Memory Variable—A temporary variable that can be referenced in any expression in reporting. It can be used in a selection expression for a data subset, in an expression that is inside a table, or in the report body, page header, footer etc. A memory variable is defined using the SET_VAR function.

Record—One complete entry in a log file. In the activity log file, for example, records consist of three fields: the time and date, label, and description for each event.

Report —A collection of historical data, real-time data, or both, organized so as to make the data easy to analyze. A report must be generated, that is, all the data must be pulled from the various sources and then organized according to the report's format. Generated reports are stored in disk files. These files can be printed as they are generated and then immediately deleted, or they can be printed later, using ControlView's LIST or PRINT commands.

Summary—A line in a table that is printed each time a key field in the data subset changes.

Table—A placeholder within the report's template used to represent the contents of a data subset. Each field in the table represents a field in the data subset. You "build" a table by identifying which fields you want included.

Template—A file that specifies both the data to be reported and the format for the report.

Introducing Reporting

Chapter 1

Creating Report Templates

Chapter 2

Defining an Expression

Chapter 3

Keys

Appendix C

Index

Chapter 1

Introducing Reporting

With the Reporting module you can generate reports of plant activity. These reports combine real time and historical data with free-form text. You can view these reports on the screen or print them on a local or network printer. You can also save them to disk.

Main Features

The main features of Reporting are:

- **You can combine data from all of ControlView's sources.** A report can contain all the real time data available in the database (not just a tag's value, but also all the tag's communication and alarm variables). To this you can add data from any of ControlView's log files: the activity log file, the alarm log file and Data Logger files.
- **You have great flexibility in extracting only the relevant data from log files.** Specific data from log files can be extracted based on complex selection criteria, and then sorted in ascending or descending order.
- **You can place security on individual reports.** Reporting is fully integrated into ControlView's security system and allows you to restrict the ability to generate confidential reports.
- **The Reporting Editor provides substantial editing power.** Besides the usual editing features such as cut, copy and paste capabilities, search and replace operations, and line drawing facilities, you can:
  - define fields (to display single values), tables (to display complete or partial log files), and expressions
  - define running headers and footers with page numbers on every page
  - insert automatic time and date stamps to show when the report was generated
  - define reports up to 132 columns wide
  - control the report's pagination
- Reports can be generated on demand with a single command or automatically with ControlView's Event Detector application module.

- You have full access to ControlView's expression language. With the expression language you have complete control over selecting and displaying data. Expressions can include:
  - values from the database or log files
  - arithmetic expressions
  - statistical functions
  - time and date functions
  - logical, bitwise, and comparison operators
  - IF-THEN-ELSE control statements
  - memory variables
- The expression language has been enhanced for Reporting: the database, Data Logger, statistical, and time and date functions, memory variables are not available in other ControlView options.

The Report Template

A template is a file that defines the format and contents of a report. You must create one template for each report you want generated.

The rest of this chapter describes the components of a template and how these components are used. The actual mechanics of using the Report Editor are described in Chapter 2, *Creating Report Templates*.

There are two components to any report template: one specifies what data to include in the report, the other determines where on the page the data is to be displayed.

Data for the Template

Data for a report can come from three different sources: the database, log files and memory variables.

The Database

The database contains two types of data, static and current value.

The static portion of the database contains information that never changes. Static information includes the tag's description, address, units, on and off labels in a digital tag, the minimum and maximum values in an analog tag, etc.

The Current Value Database (or CVD) is the portion of the database that contains the values that are always changing. Besides the tag's value, the CVD stores the tag's communication status (i.e., whether a communication error has occurred while scanning the tag) and all the alarm information for a tag (if the Alarming option is installed).

Log Files

Reporting can retrieve data from ControlView's log files. There are three types of log files:

- the alarm log file
- the activity log file
- the log files created by the various Data Logger models

Rather than extracting data directly from any of these log files, Reporting moves the data to temporary files called "data subsets". Data subsets give you added flexibility in specifying what data to use in a report.

In its simplest form, a data subset will contain the entire contents of a log file. Usually, however, the data subset will be a portion of the original file. For example, to create a report on the communication network's performance, you would define a data subset that includes only the communication activity from the activity log file.

To define data subsets based on the activity log or alarm log files, you must be familiar with the record structure of these files. (A *record* is one entry in the file. A record may contain several tags. In the activity log file, for example, each record contains the time, date, label and description of each event that was recorded. Each element within a record—the time and date, label and description in the case of the activity log—is called a *field*). Appendix B, *File Formats*, shows the format for the activity log file, and the default alarm log file.

The file format for Data Logger files is complex, since each record contains a snapshot of all the tag values for that Data Logger model. Accordingly, Reporting provides a number of functions used to define data subsets based on Data Logger files.

There are two operations you can perform on a log file when you define a data subset. You can:

- select certain records. This operation allows you to filter out unwanted data. For example, you can define a subset of the alarm log file that contains only alarm records for the highest priority alarms. Expressions are used to select data.

- sort the records within the subset. For example, the records in the alarm log are stored chronologically. You could sort the contents by tag name so all the alarms for one tag are grouped together.

Important: Only the activity log or alarm log files can be sorted; Data Logger models cannot be sorted.

Important: To define complex data subsets you can specify one data subset as the source in a subsequent data subset. In this way you can cascade data subset definitions, so that one definition further refines the contents of an existing data subset.

Important: A data log subset cannot be used as a source for any subsequent subsets.

Memory Variables

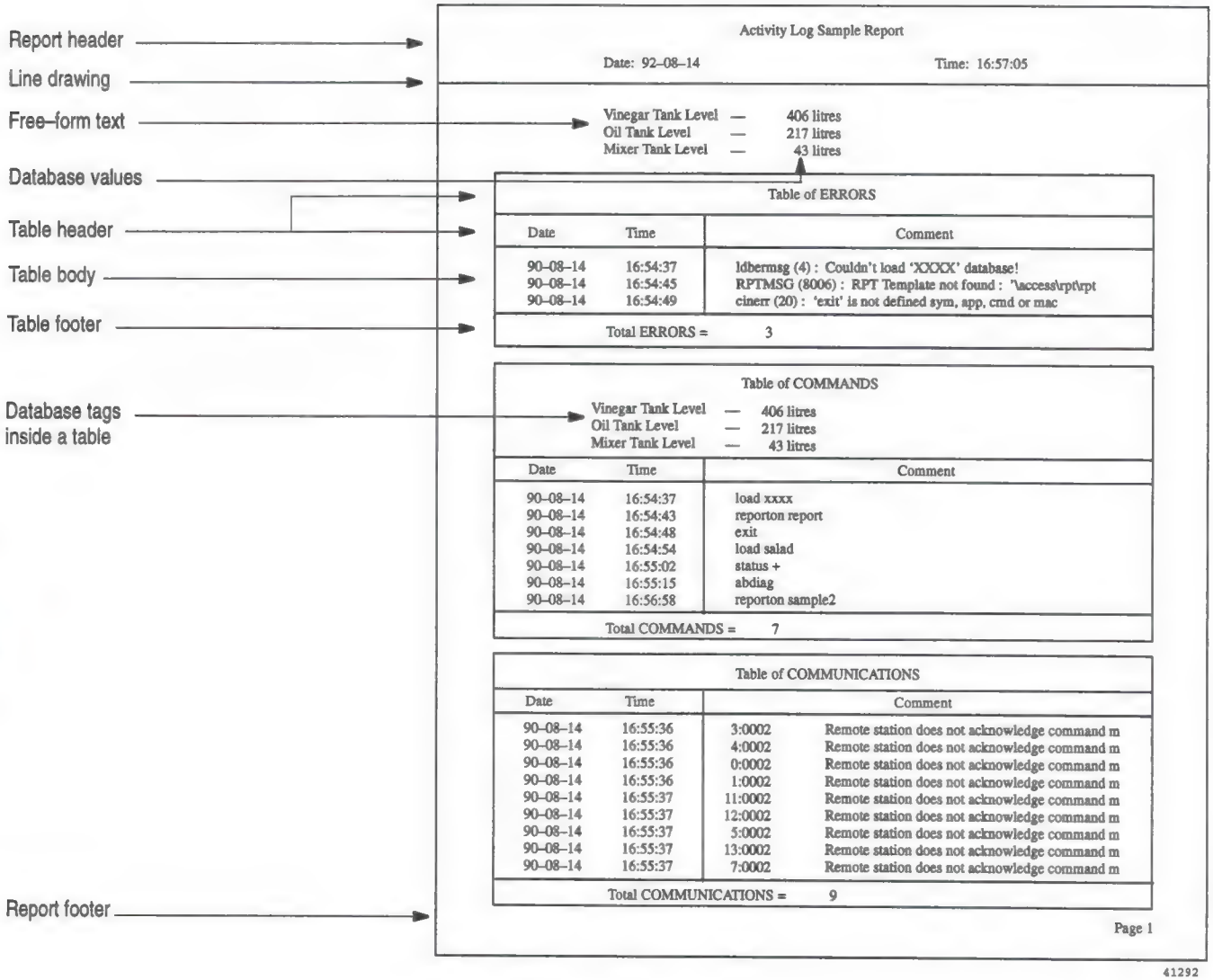
The SET_VAR function is an advanced feature enabling you to define memory variables. A memory variable can be referenced anywhere a tag name or data subset field name can be. Inside a table, Reporting will check any identifier, first to see if it matches a field in the data subset, then to see if it is a tag in the CVD, and finally, to see if it is a memory variable. Outside a table, Reporting checks the CVD then the memory variables.

The Report Layout

A basic report will likely consist of three components: free-form text, tables and fields. You can create more sophisticated reports that include:

- summary lines within the table
- running headers and footers
- horizontal and vertical lines separating different portions of the report
- automatic time and date stamps
- forced page breaks

Figure 1.1
Components of a Report



Free–form Text

The Report Editor allows you to place text anywhere on the screen and print that text as part of the report. You can use free-form text to add labels or column titles, paragraphs, or even entire covering pages to a report.

Tables

A table allows you to print the contents of a data subset. When you define a table, you must specify which data subset to use. You then build the table by identifying which fields you want printed.

When the report is generated, Reporting reads through the data subset one record at a time and prints the values for the specified fields. The length of the table will depend on the size of the data subset. A single table may span numerous pages. You can add table headers and footers when you define the table. Table headers are printed at the top of the table, and at the top of each page if the table spills over onto subsequent pages. Similarly, the table footer is printed at the end of the table, and before a page break if the table extends onto the next page.

You can also add table summaries. A table summary is useful only when the subset is in sequence—either in chronological order or after it has been sorted. You then specify the key field (the field that controls the sequence of the data subset). Whenever that field changes, a summary line will be printed so that the summary lines are mixed with the lines of the table. The summary line flags a change in the key field. For example, if the data subset is sorted by tag name, then whenever the tag name changes, a summary line is printed.

Fields

Fields are used in two different ways within a report template depending on whether they are defined inside or outside a table.

Outside a table, for example in the report's body, header or footer, a field refers to a tag in the database or a memory variable.

Inside a table, the field refers to one of these:

- specific item in each record of the data subset
- tag in the database
- memory variable

Each field will represent a column in the table, while each record in the data subset will correspond to one row.

Creating Reports

The basic steps for creating a report are:

1. Call up the Report Template Browser. Either choose *Edit Report Template* under Configure in the Setup Menu (see Chapter 2, *Creating Report Templates*), or use the REPORT command from the command line (see Appendix A, *Reporting Commands*).

2. The first time you call up the Report Template Browser, you may wish to change Reporting's default settings for page size, system directories, and printer settings. You make these changes by choosing *Setup* from the Report Template Browser menu. See Chapter 2, *Creating Report Templates* for details.

   You can also change any of these settings for individual templates. You do this as part of defining the template.

3. Create a new template: choose *Create* from the Report Template Browser menu. You'll be asked to enter a name and description for the report. See Chapter 2, *Creating Report Templates* for details.

4. Define the data subset, i.e., specify which log files you want to use, and how you want to manipulate the data. If the report is limited to real-time data from the Current Value Database, no data subsets need to be defined. However, if the report will display historical data, then the data subsets have to be defined before the report template can be completed.

   To define a data subset, choose *Data Select* from the Report Editor's menu. For details, see Chapter 2, *Creating Report Templates* and Chapter 3, *Defining an Expression.*

5. You should already have defined default settings for all reports back in step 2. However, you may want a particular report to differ from the default settings when it comes to the size of the printed page, whether the report is to be printed, saved to disk or both, etc. To define the unique characteristics for the report, choose *Configurations* from the Report Editor's menu. See Chapter 2, *Creating Report Templates* for details.

6. Finally, you must define the actual report. This involves defining any headers and footers, adding free-form text and line drawings, and defining all the fields, tables, and summaries that will display ControlView data.

   The mechanics of using the editor is explained in Chapter 2, *Creating Report Templates*. You will also need to read Chapter 3, *Defining an Expression*, before you can define fields.

7. Save the report template by choosing *Save and Exit* from the Report Editor's File menu.

8. Generate the report. Either choose *Generate Report* under Tools in the Actions Menu (see Chapter 4, *Generating Reports*), or by using the REPORTON command (see Appendix A, *Reporting Commands*).

The Sample Reports

Included on the Reporting disk are two sample reports, called SAMPLE1 and SAMPLE2. You may want to call up the Report Editor and examine these reports and then try generating them.

To call up the Report Editor, choose *Edit Report Template* under Configure in the Setup Menu and then choose the report from the list.

SAMPLE1

The report SAMPLE1 is a simple report. It contains a report header, footer, free-form text, and fields. It displays real-time data from the SALAD database. This report does not use any logged data, and therefore does not have any tables.

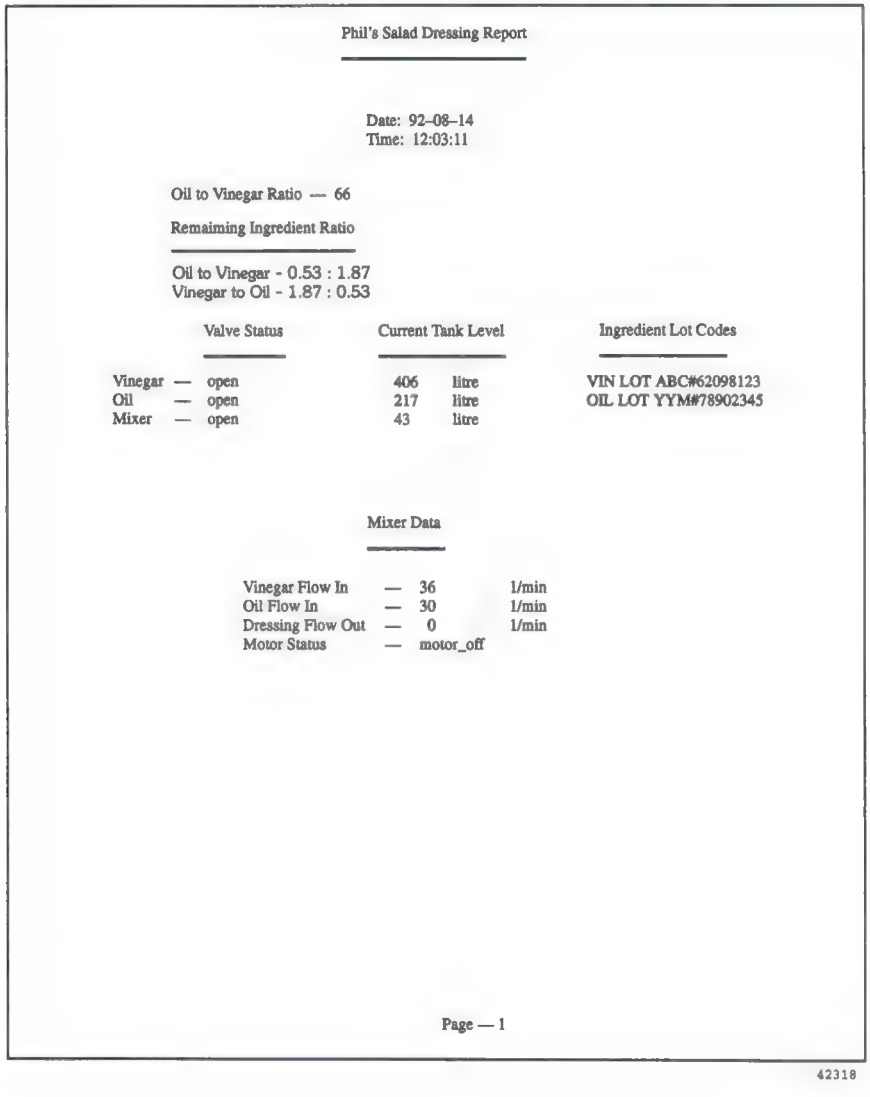
To generate this report:

1. Load the SALAD database: choose *Load Database* under Load in the Actions menu and then choose *SALAD* from the list.

2. Generate the report: choose *Generate Report* under Tools in the Actions menu and then choose *SAMPLE1* from the list.

3. View the report: the report will be displayed on the screen automatically as soon as it is generated. To view the report at a later date: choose *List File* under tools in the Actions menu. When prompted to name the file you want to list, where C is the drive where ControlView is installed, type:
   `C:\access\rpt\out\sample1` *press Enter*

Figure 1.2
The Sample1 Report



Phil's Salad Dressing Report

Date: 92–08–14
Time: 12:03:11

Oil to Vinegar Ratio — 66

Remaiming Ingredient Ratio

Oil to Vinegar - 0.53 : 1.87
Vinegar to Oil - 1.87 : 0.53

	Valve Status	Current Tank Level		Ingredient Lot Codes
Vinegar	— open	406	litre	VIN LOT ABC#62098123
Oil	— open	217	litre	OIL LOT YYM#78902345
Mixer	— open	43	litre	

Mixer Data

Vinegar Flow In	—	36	l/min
Oil Flow In	—	30	l/min
Dressing Flow Out	—	0	l/min
Motor Status	—	motor_off	

Page — 1

42318

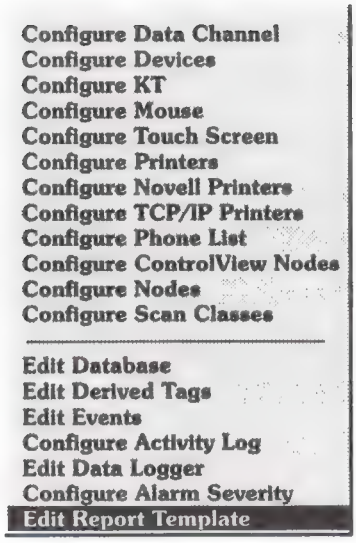
SAMPLE2

The report called SAMPLE2 is a more complex report. Besides using headers, footers, free-form text and fields, this report uses three tables. These tables display data from the activity log file. The first table shows all the logged error messages; the second shows all the logged operator and internal commands; the third table shows all the logged communication activity. Each table has a table footer displaying the total number of entries in that table.

Figure 1.1 is an illustration of the report SAMPLE2.

The two reports, SAMPLE1 and SAMPLE2, are included for you to examine. It's unlikely that either report would be used in a real plant-floor environment; their purpose is to illustrate certain features of reporting. For example, the SAMPLE2 report defines four data subsets. The exact same report could have been defined using only three data subsets. Four were used to illustrate how one data subset can be used as a source of data for a second data subset.

SAMPLE2 and Data Subsets

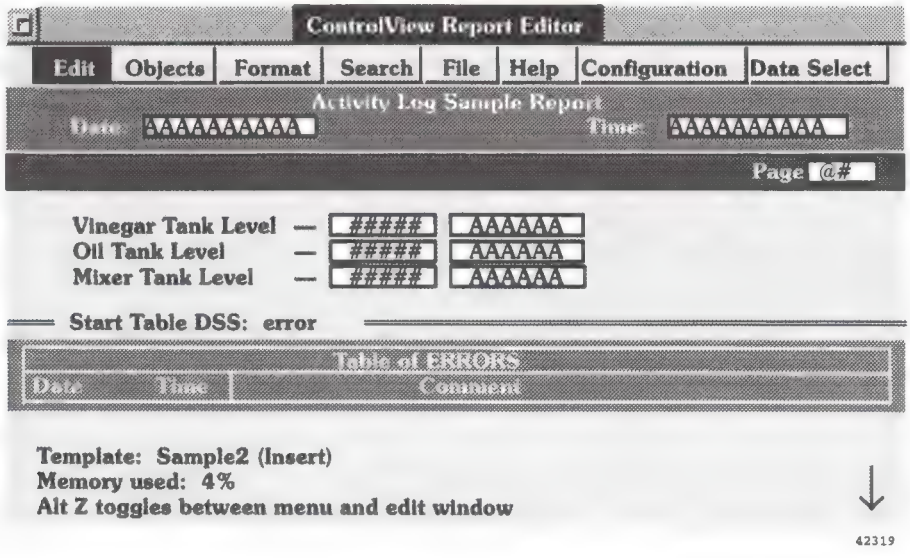


To better understand the following discussion on the four data subsets, load the report and examine the data subset definitions. To do this:

1. Load the Report Editor with the SAMPLE2 template: Choose *Edit Report Template* under Configure in the Setup Menu. From the list of templates, choose *SAMPLE2*.

   The Report Editor loads and the SAMPLE2 template is displayed on the screen.

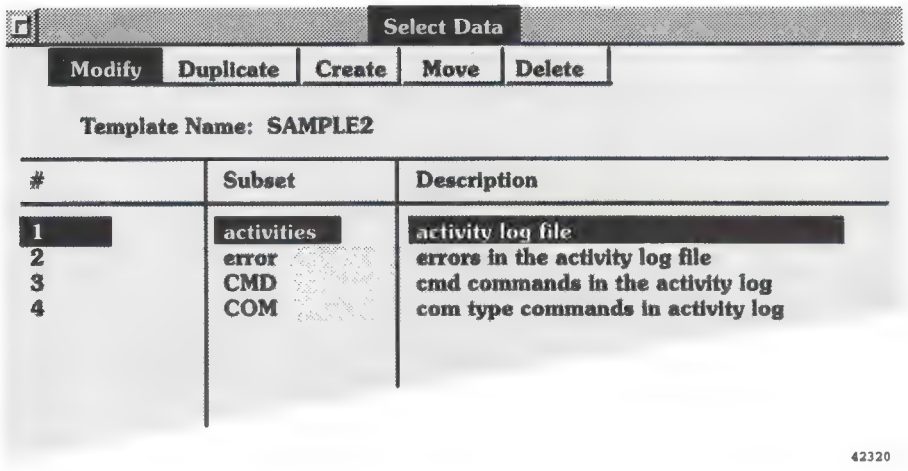
Figure 1.3
SAMPLE2 As It Appears On The Screen



2. Call up the list of data subset definitions: Choose *Data Select* from the Report Editor's menu.

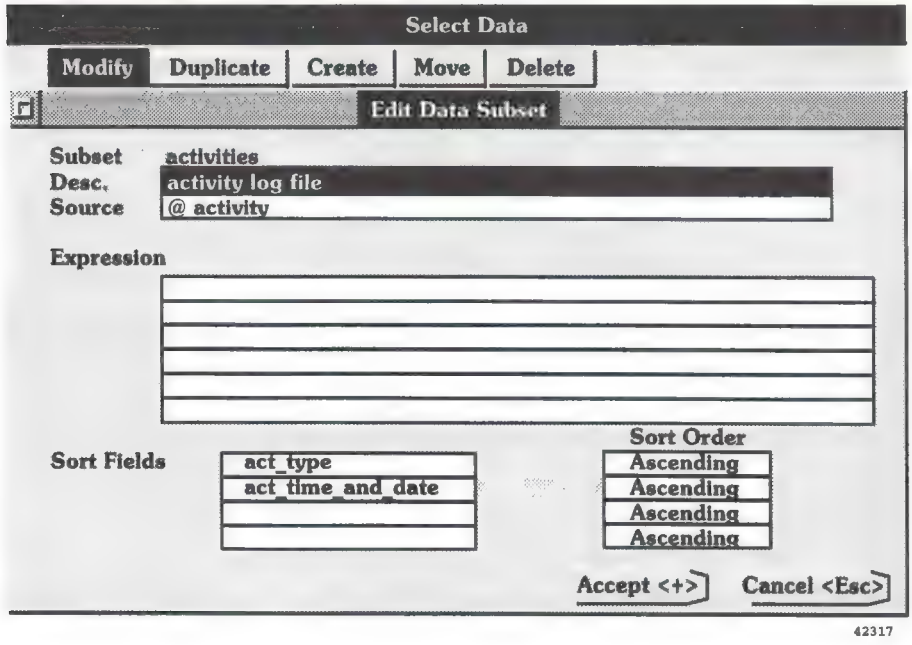
   The Select Data screen appears, listing the four data subsets.

Figure 1.4
The Select Data Screen



3. Examine each data subset definition: Highlight the definition and choose *Modify*.

Figure 1.5
Definition of the ACTIVITIES Data Subset



Pressing **Esc** will close the current screen. You can press **Esc** once to close the Edit Data Subset screen, or press it a number of times to leave reporting entirely.

The first data subset in the list, ACTIVITIES, is the entire activity log file sorted first by activity type, and then by time. Unlike the other three data subsets, ACTIVITIES is not used to build a table. Instead, it is used as a source for the three other data subsets.

If you look at the definitions for the three remaining data subsets, ERROR, CMD, or COM, you'll see that:

- the *Source* field specifies ACTIVITIES, the first data subset, not @ACTIVITY, the activity log file
- the *Expression* field is used to select part of the data
- the *Sort Fields* are blank

By specifying ACTIVITIES instead of the activity log file, there is no need to sort the data, since ACTIVITIES is already sorted.

SAMPLE2 could be defined with only three data subsets, ERROR, CMD, and COM. To accomplish this, specify the activity log file as the data source for each data subset (instead of the ACTIVITIES data subset), and then sort each by activity type and date.

Sorting is the slowest part of generating a report. Accordingly, sorting the log file before extracting the data would be faster only if the log file is small and each data subset comparatively large. However, if the log file is big and the data subsets comparatively small, then extracting the data first and then sorting each subset could prove to be faster. The general rule is to sort the smallest amount of data possible to define all the necessary subsets. Sorting also has an impact on memory utilization.

Restrictions

There are two restrictions on a template: the number of tags that can be referenced and the amount of computer memory that can be used.

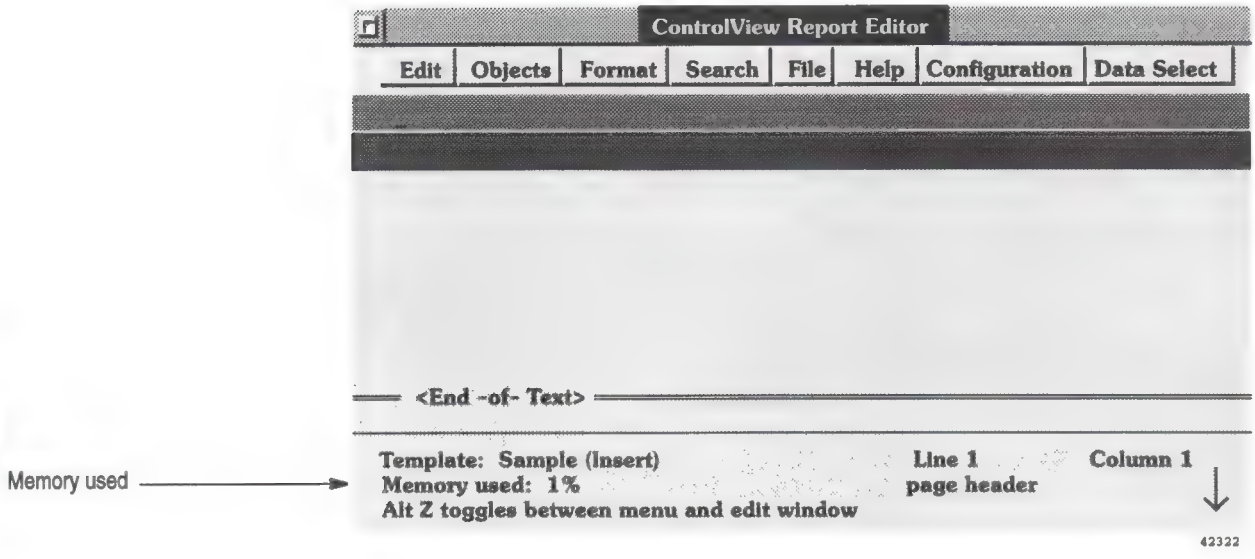
Maximum Number of Tags

A report template can reference up to 500 tags. Each tag can be used in expressions any number of times; the restriction is on the total number of unique tags specified.

Maximum Memory Used

All text, field definitions, expressions, data subset definitions, table definitions, etc., consume computer memory. The Report Editor calculates the percentage of available memory the template has consumed. This figure is displayed near the bottom of the editor's screen.

Figure 1.6
The "Memory Used" Display



A template that has used 100% of available memory may be too large to run. That's why the editor warns you when the memory becomes 90% full.

Important: If you see this warning and still have more to add, consider breaking the report into two templates. Failure to do this may cause the template to be lost.

Chapter 2

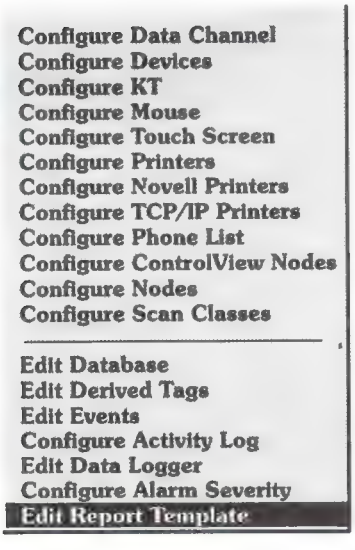
Creating Report Templates

This chapter describes the features of the Report Editor and the mechanics of how to use them.

The Report Editor is designed around two principal windows: the Report Template Browser and the Report Editor. This chapter is divided into two sections, the first for the Browser and the second for the Editor. Each section follows the sequence of the menus presented in that window.

The Report Template Browser

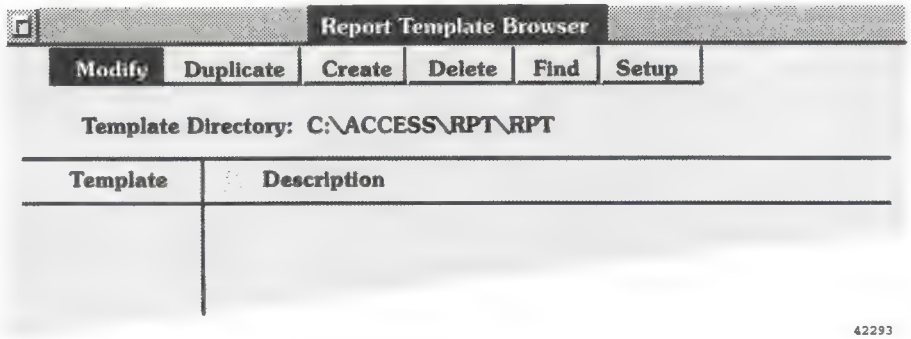
The Report Template Browser is the first level of the Report Editor. From this window, you create, delete, and modify templates. You also use the Setup menu option to define default values for the reports, defaults that can be overridden for individual reports.



To create a new template or call up the list of existing templates, choose *Edit Report Template* under Configure in the Setup Menu.

A list of existing report templates pops up. If you choose one of these templates, the Report Editor loads and the template is displayed. If, on the other hand, you choose *new*, the following window appears.

Figure 2.1
The Report Template Browser Window



Modifying a Template

To edit an existing report template:

1. Highlight the desired template in the list.

2. Choose *Modify* from the Report Template Browser menu.

ControlView loads the Report Editor with the specified template. In Report Editor you specify the layout of the report and define the data subsets. Details on using the Report Editor appear later in this chapter.

Duplicating a Template

To make a copy of a template, complete with data subset definitions:

1. Highlight the desired template in the list.
2. Choose *Duplicate* from the Report Template Browser menu.
3. When prompted, type the name for the duplicate report template.

ControlView makes a copy of the specified template.

Creating a Template

To create a new report template:

1. Choose *Create* from the Report Template Browser menu.
2. When prompted, type the template's name and description. The name can be up to eight characters long and can use letters, numbers, and the dash (–) and underscore (_).

ControlView loads the Report Editor with an empty template. You then define the data subsets and specify the layout of the report. Details on using the Report Editor appear later in this chapter.

Deleting a Template

To delete a template:

1. Highlight the template to be deleted.
2. Choose *Delete* from the Report Template Browser menu.
3. When prompted, confirm that you do want to delete the specified report template.

Finding a Template

If the list of templates is long, it may be easier to locate a specific entry using the *Find* menu option than to scroll through the list. To find an entry quickly:

1. Choose *Find* from the Report Template Browser menu.

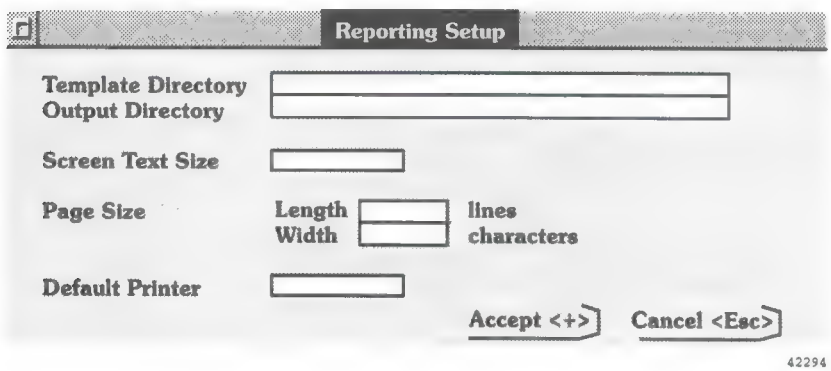
2. When prompted, type in enough letters of the template name to identify it, and press **Enter**.

The specified template will be highlighted. If the template does not exist, the template with the most similar name will be highlighted.

Setup

The Setup option allows you to define default values for report templates. The directories, page size and printer settings can all be changed for individual templates. When you choose *Setup* the following window appears:

Figure 2.2
The Reporting Setup Window



- Template Directory

  The *Template Directory* field names the drive and directory where Reporting should save template files. The default is C:\ACCESS\RPT\RPT where C is the drive where ControlView is installed.

Important: Whenever you configure a pathname, be sure to start with the drive letter. This is absolutely essential when running ControlView in a multi-drive environment.

- Output Directory

  When Reporting generates a report, it writes the information to a disk file. You can design the template so that Reporting will immediately send the file to the printer and delete the disk file. Alternately, you can save the disk file and later view the report on the screen with the LIST command, or print it out with the PRINT command.

  The *Output Directory* field specifies the drive and directory where Reporting will place the generated report file. The default is C:\ACCESS\RPT\OUT where C is the drive where ControlView is installed.

Important: Whenever you configure a pathname, be sure to start with the drive letter. This is absolutely essential when running ControlView in a multi-drive environment.

- Screen Text Size

  The *Screen Text Size* determines how many lines of text can be displayed on the screen.

  Options are:

  - Small — 31 lines
  - Normal — 18 lines (the default)

- Page Size

  Setting the length of the page determines how many lines are printed before ControlView sends a form feed command to the printer. Setting the page width determines the number of characters that will be printed on each line.

  The page length ranges from 0 – 256, where 0 means do not generate form feeds. The default is 60. The page width can be as narrow as 40 and as wide as 132 characters, with the default width of 80.

- Default Printer

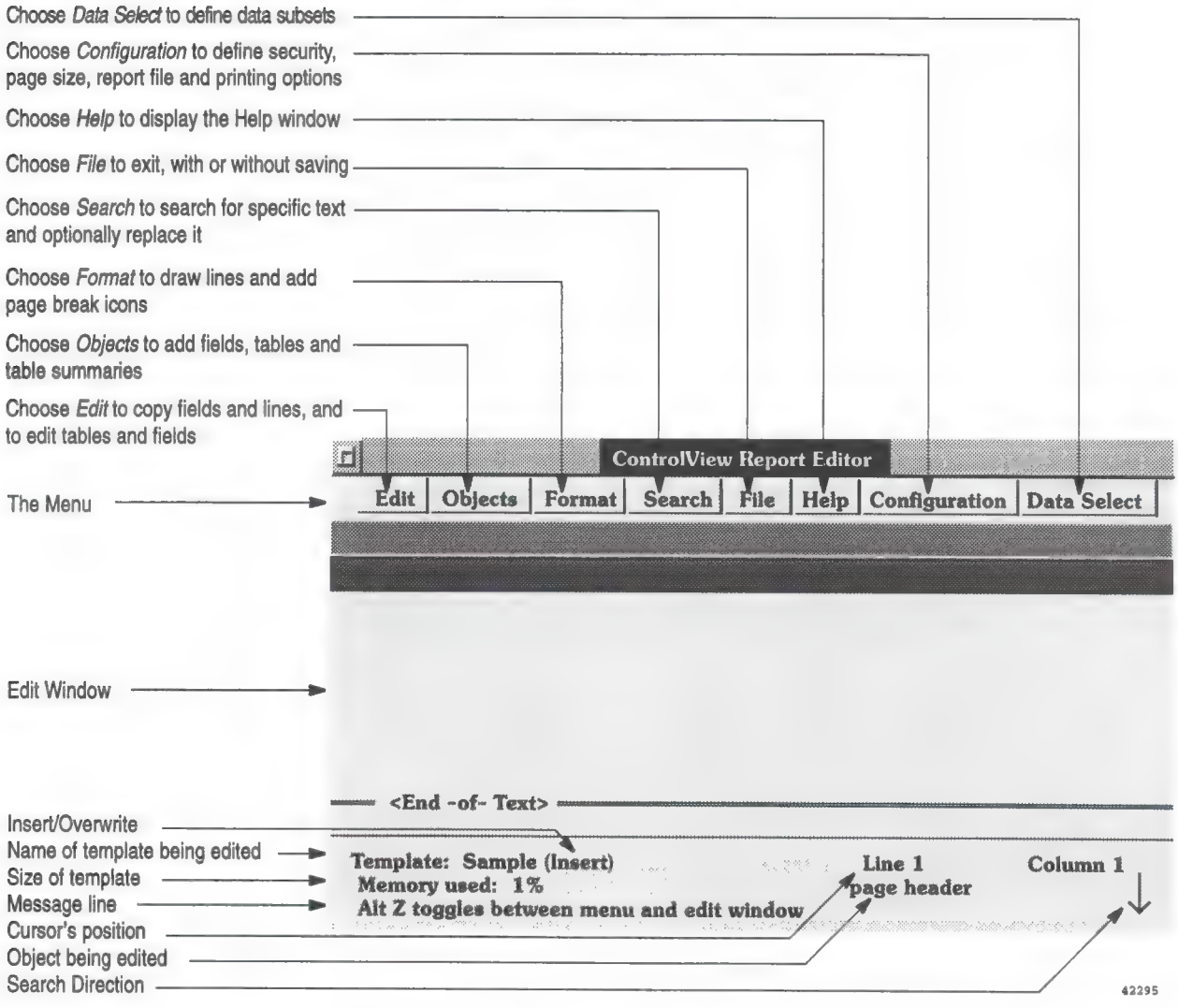
  The *Default Printer* field specifies which printer the report should be printed on (if it is to be printed at all). Choices are Printer1, Printer2, Printer3, and Printer4, with the default being Printer1.

The Report Editor

This section of the chapter describes the Report Editor. In the Report Editor you actually define a template, i.e. define the tables and fields, and define the data subsets.

To call up the editor, choose either *Create* or *Modify* from the Browser menu. The following window appears:

Figure 2.3
The Report Editor Window



Notice the message line at the bottom of this window. The Report Editor displays useful information here.

Many of the menu entries also have keyboard short cuts. For example, to begin drawing a line, choose *Line Drawing* from the Format menu, or press **Alt-D**. See Appendix C, *Keys* for details.

Pressing **Alt-Z** moves the cursor from the menu to the edit window. Pressing **Alt-Z** again returns the cursor to the menu.

Once you've created a new template, and have filled in its name and description, you are presented with a blank Report Editor window. To build up the template, follow these steps:

1. If necessary, change the security setting, file options, page size, and printing option located in the *Configuration* menu.

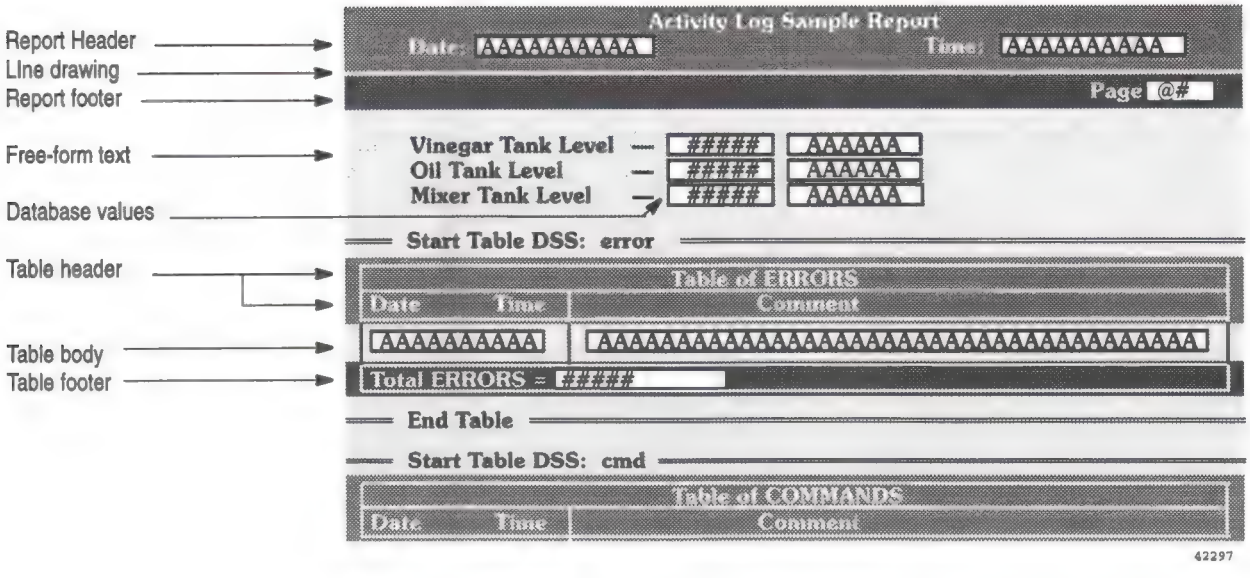
2. Choose *Data Select* and define the data subsets if the report will use data from any of ControlView's log files.

3. Move the cursor into the edit window and define the report layout.

4. Save the template to disk.

Figure 2.4 shows a report as it would appear in the Edit window.

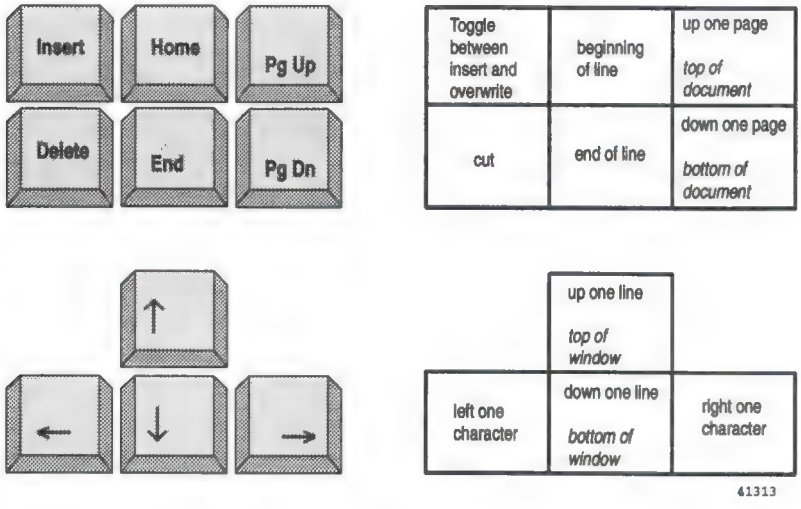
Figure 2.4
Components Of A Report As They Appear On The Screen



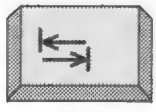
Editing Keys

The cursor control keys on the numeric keypad are used to move the cursor about on the editing window, as illustrated in Figure 2.5.

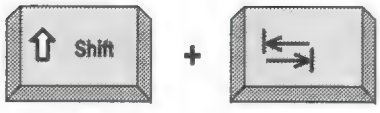
Figure 2.5
The Cursor Control Keypad



Several other keys are important for editing:

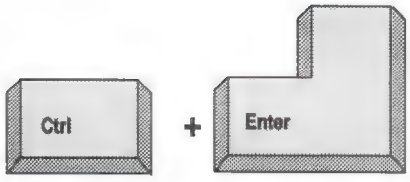


(Tab) move cursor right eight characters

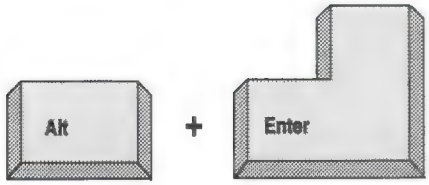


move cursor left eight characters

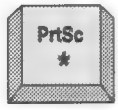
toggle between insert and overwrite modes



insert a new line above the cursor



insert a new line below the cursor



delete from cursor to end of line

delete entire line that cursor is on. The deleted line is copied to the buffer—overwriting the buffer's contents—and can be inserted elsewhere in the template

Screen Colors

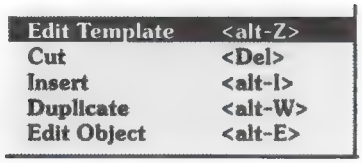
Different parts of the report appear in different colors on the screen.

Table 2.A
Colors Used in the Edit Window

This item:	has these colors:
text	black on grey
fields	black on cyan
headers	cyan on blue
footers	cyan on navy blue

The Edit Menu

The *Edit* menu allows you to move into the edit window, copy fields and lines, and edit tables and fields.



Edit Template

Choose *Edit Template* to move the cursor into the edit window.

When the cursor is in the window, you can add or change text, move the cursor about, and-with the keyboard short cuts-add or change tables, fields, and other items found in the menus.

The **Alt–Z** key combination is a toggle that takes you to the menu from the edit window, or to the edit window from the menu.

Cut

Choose *Cut* (or press **Del**) to remove a character, field, or icon from the template.

Important: If you cut a field, it is saved to a temporary buffer - replacing the previous contents of the buffer - and can then be inserted elsewhere in the report. A complete line can be cut to the buffer using the delete line key (the keypad minus key).

Table 2.B
The Cut Operation

When the cursor is:	choosing *Cut* (or pressing Del)
on a character in free-form text	removes the character
on a field	removes the field and stores it in a buffer so that it can be inserted elsewhere
on an icon (for a table, summary, or hard page break)	removes the table, summary, or page break

Insert

Choose *Insert* (or press **Alt-I**) to place the contents of the buffer - either a field or line - into the report at the cursor's location.

When you insert a line from the buffer, it is placed below the cursor's location.

When you insert a field in the middle of a line, the remainder of the line is shifted to the right to make room for the field.

Important: In Insert Mode, if you try to insert a field and there is not enough room on the line to accommodate the field, the insert operation will not work. In Overwrite Mode, text may be overwritten and lost.

Duplicate

Choose *Duplicate* (or press **Alt-W**) to copy the line or field to the buffer while leaving the original line or field intact on the screen. The copy can be inserted elsewhere in the report.

If the cursor is in a field when you choose Duplicate, the field is copied to the buffer; otherwise the entire line (with any fields in that line) is copied to the buffer.

Important: When you duplicate a line or field, it will replace the previous contents of the buffer.

Edit Object

Choose *Edit Object* (or press **Alt-E**) to modify a field's definition or change the data subset for a table.

The cursor must be:

- inside the field to edit the field's definition
- on the table icon to specify a different data subset for the table
- on the summary icon to specify a different trigger field for the summary

The Objects Menu

The elements in the *Objects* menu allow you to add fields, tables, and table summaries.

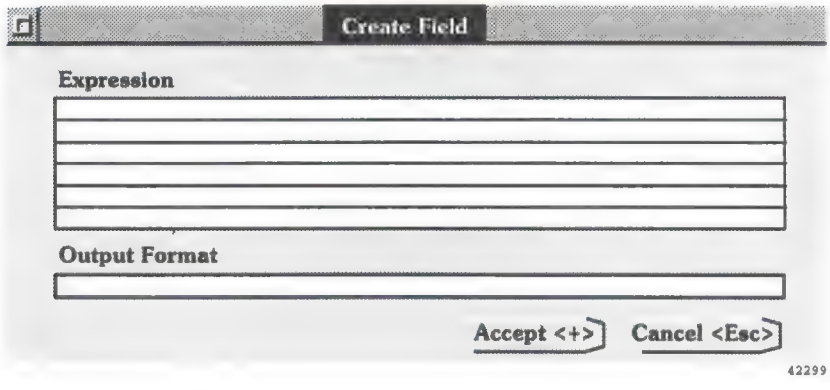
Field

Field	<alt-F>
Table	<alt-T>
Summary	<alt-U>

Choose *Field* (or press **Alt-F**) to place a field into the report at the cursor's location. If the field is located between the start and end table icons, the field's value will be extracted from the data subset specified for that table, or the CVD or the memory variables. If the field is located outside a table, the field's value will come from the CVD or the memory variables.

When you create a field, the *Create Field* window appears.

Figure 2.6
Create Field Window



- *Expression* Type in an expression defining the value to be displayed in the report. In its simplest form, the expression is the name of a tag or a data subset field. However, expressions can be complex mathematical equations that convert one or more tag or data subset values into meaningful information. For details on defining an expression, refer to Chapter 3, *Defining an Expression*.

- *Output Format* Specify the format Reporting should use to display the data. The format string identifies the type of data (character versus number) and the length of the field, as well as the actual data format (whether the data is left or right aligned, whether numeric data has leading zeros or spaces, or has plus and minus signs).

  The *Expression* and *Output Format* fields are related. An expression can return either a number or text string, and the *Output Format* must specify which data type to expect.

Table 2.C
Formatting Characters

Character	Meaning	Example
A	text field	AAAAAAAAA
#	numeric field	#######
.	decimal point	###.####
,	comma	#,###
( )	parentheses with format character outside and a number inside that repeats the format character the specified number of times	#(5) is the same as ##### A(5) is the same as AAAAA
( )	parentheses with format characters inside to display negative numbers in parentheses	(###)
@	left justification moves data to left end of field (numbers are normally right justified)	@##### @AAAAA
~	right justification moves data to right end of field (text is normally left justified)	~##### ~AAAAA
^	null justification removes spaces before and after the field	^AAAAA ^#####
0	zero fill pads the beginning of the number with zeros	0####
$	floating dollar sign places dollar sign immediately before the digits	$####
+	minus/plus signed field places a plus or minus sign at the front of the field	+#####
–	minus/blank signed field places a minus sign in front of negative numbers	–#####

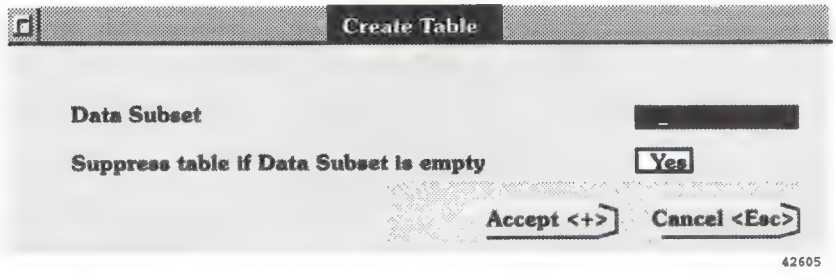
Table 2.D
Sample Format Definitions

This value	into this format definition	means this	and is printed like this
0	#####	numeric field	0
-12345	#####	five numeric characters, won't display five digits and sign *	?????
123456	#####	five numeric characters, won't display a six digit number *	?????
1234	#,###	numeric field with comma	1,234
1234	####.##	numeric field with 2 decimal places	1234.00
12.34	###.##	numeric field with 2 decimal places	12.34
12.349	##.##	numeric field with 2 decimal places-decimals are rounded	12.35
12.349	#.###	numeric field is too small *	?????
12	0####	zero padding	00012
12	$####	floating dollar sign	$12
12	+####	signed field	+12
123	-####	signed field	123
-123	(####)	numeric field in parentheses	(123)
12	@####	left justified numeric field	12
"PLANT1.TAG1"	A(15)	15 characters	PLANT1.TAG1
"PLANT1.TAG1"	AAAAA	5 characters	PLANT
"PLANT"	~A(15)	5 characters, right justified	PLANT
"PLANT1.TAG1"	~A(15)	15 characters, right justified	PLANT1.TAG1
* An error message will appear in the generated report.			

Table

Choose *Table* (or press **Alt-T**) to place a table in the report at the cursor's location.

Figure 2.7
Suppressing Empty Tables

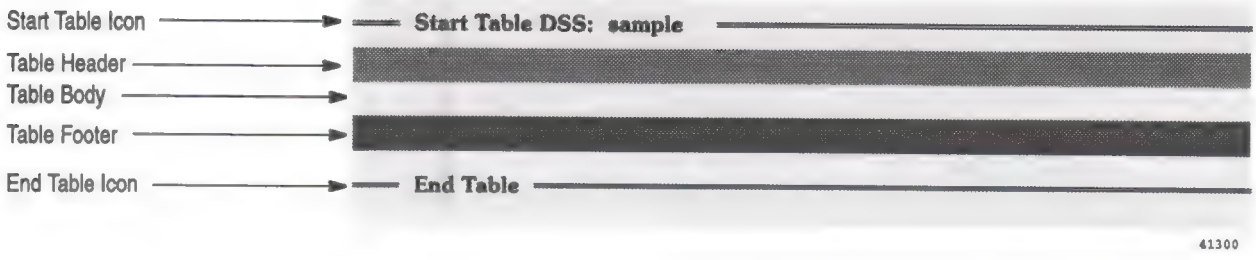


You are asked to identify the data subset for the table. You are also asked whether you want to suppress the appearance of the table in the report, when the data set is empty.

Important: Memory variables calculated in a suppressed table will not be set.

After you choose *Accept,* a table icon is displayed on the screen.

Figure 2.8
The Table Icon



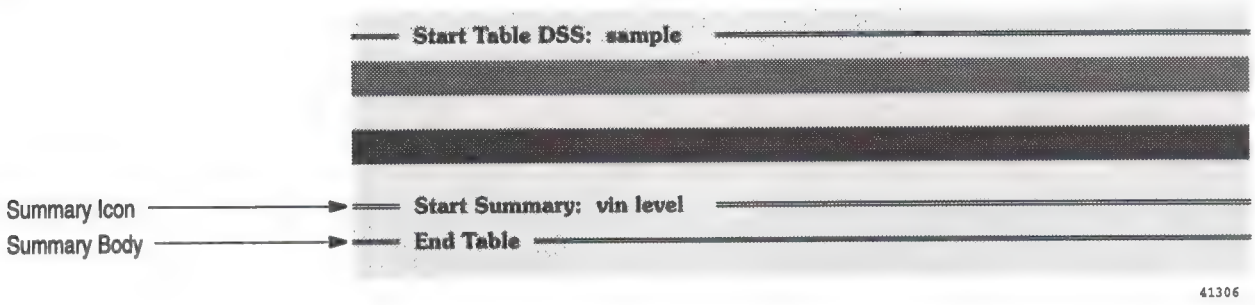
Add the fields in the table body. The fields will extract their data from the data subset identified for the table or from tags in the database or from memory variables. You can also add free-form text to the table body, but that text will appear in every row in the table.

Summary

A summary is a line that is printed within a table every time the key field changes. The key field is the field that determines the order of the data subset. For example, if the alarm log file has been sorted by tag name, then the ALM_TAG_NAME is the key field. If the data subset is in random order, then it has no key field and defining a summary becomes meaningless.

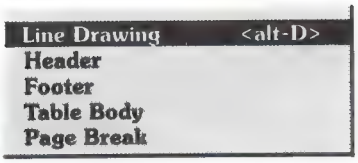
To define a summary, place the cursor inside a table, and choose *Summary* (or press **Alt-U**). You are asked to identify the key field for the summary. After you've typed the name of the field, a summary icon is displayed inside the table icon.

Figure 2.9
The Summary Icon



In the space between the Summary and Table End icons, enter any text or line drawings or fields you want displayed on the summary line.

The Format Menu



The *Format* menu allows you to enter line drawing mode, and add page break icons.

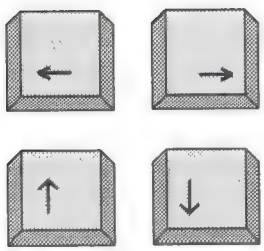
With this menu you can also add page or table headers and footers and the body of a table. You won't normally need to add any of these items to your template; they are present when you create a new template or add a new table. You can increase the size of an existing header, footer, or table body by moving in the cursor, then pressing **Enter**.

However it is possible to delete a header, footer, or table body, so you'd choose these menu items to put them back in the template.

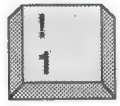
Line Drawing

Choose *Line Drawing* (or press **Alt-D**) to add lines to a report.

These are the keys that are used for drawing lines:



draw the line



specify single line mode

specify double line mode

specify erase mode. Use the arrow keys to backtrack over the existing line and erase it.

specify move mode. Use the arrow keys to move without drawing a line.

end line mode

Header

Page headers are automatically placed at the beginning of a new template and table headers are automatically placed at the beginning of a new table. You only need to add a header if you have previously deleted it.

A table header may contain text, line drawings, Page#, time and date functions, database functions, references to memory variables, and tags in the Current Values Database. A table header may *not* contain statistical functions, fields from the data subset or functions related to data logger subsets.

Choose *Header* to add a page or table header. If the header already exists choosing *Header* will add another line to the header.

Footer

Page footers are automatically placed at the beginning of a new template and table footers are automatically placed at the end of a new table. You only need to add a footer if you have previously deleted it.

A table footer may contain text, line drawings, Page#, time and date functions, database functions, references to memory variables, tags in the Current Values Database, and statistical functions. A table footer may *not* contain fields from the data subset or data logger file access functions.

Choose *Footer* to add a page or table footer. If the footer already exists choosing *Footer* will add another line to the footer.

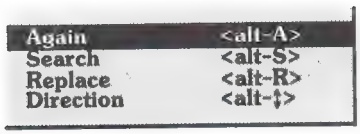
Table Body

A newly created table will have one blank line as the table body. You may not want a table body, if, for example, you want to generate only a footer containing statistical functions. If you delete the table body and then decide you want it back, make sure the cursor is inside the table and choose *Table Body.*

Page Break

To force a page break in the printed report, move the cursor to the location where you want the page break to occur and choose *Page Break.* A double line represents the page break on the screen. The printer will skip to a new page each time it encounters a page break icon in the report.

The Search Menu



The *Search* menu allows you to search for specific text within a report, and if desired, replace the text with different text.

Again

Choose *Again* (or press **Alt-A**) to repeat the previous search operation.

Search

To search for a specific item of text, choose *Search* (or press **Alt-S**) and specify the text you want to find. The search begins at the cursor's location and continues to the beginning or end of the document, depending on the direction of the search.

The direction of the search is indicated by the arrow at the lower right hand corner of the screen. Choose *Direction* (or press the **Alt** key and the up or down arrow keys) to change the direction of the search.

Important: The search operation is case sensitive, so you must type upper and lower case exactly as they appear in the template.

Replace

To search for a specific item of text and replace it with different text, choose *Replace* (or press **Alt-R**). You'll be asked to type in the text you want to search for and the text you want to replace it with.

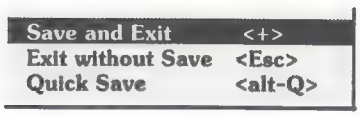
The direction of the search is indicated by the arrow at the lower right hand corner of the screen. Choose *Direction* (or press the **Alt** key and the up or down arrow keys) to change the direction of the search.

Important: The replace operation is case sensitive, so you must type upper and lower case exactly as they appear in the template.

Direction

The direction of the search or search and replace operation is indicated at the bottom right hand corner of the screen. To change the search direction, choose *Direction* or press the **Alt** key and the up or down arrow key.

The File Menu



The *File* menu allows you to save or abandon changes you've made to a template.

Save and Exit

Choose *Save and Exit* (or press the + key) to save the template to disk and exit the editor. You return to the Report Browser window.

Exit without Save

Choose *Exit without Save* (or press **Esc**) to abandon any changes you've made to the template. You are asked to confirm that you want to exit without saving. Press the + key to confirm and you return to the Report Browser window.

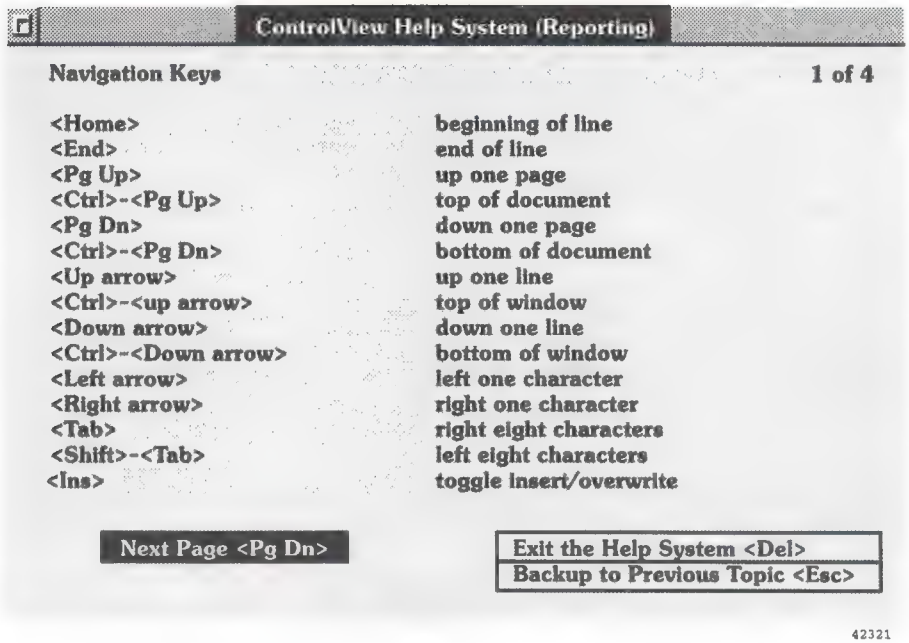
Quick Save

Choose *Quick Save* (or press **Alt-Q**) to save the template and continue working.

The Help Menu

Choose *Help* (or press **Alt-H**) to call up help windows. These help windows are a quick reminder of the keys used in the editor.

Figure 2.10
The First Help Window

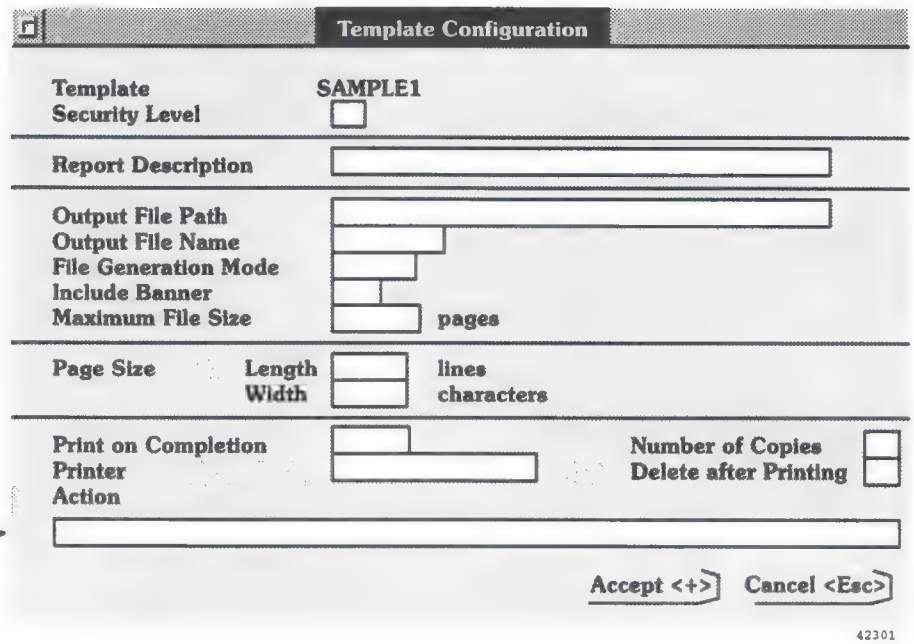


The Configuration Menu

Choose *Configuration* to call up the Template Configuration window. From this window you set parameters for the individual template, including:

- security
- report file options
- page size
- printing options
- command to be run when the report is completed

Figure 2.11
The Template Configuration Window



- Security Level

  ControlView security is covered in detail in Chapter 2, *The Setup Menu* of the *ControlView Core User Manual*. There are sixteen access codes (A through P) and "no security" code, the asterisk.

  The asterisk access code means that any user, even one that hasn't signed onto the system, can use that command, macro, tag, display, or template.

  The asterisk is the default code for Reporting templates. To restrict access to generating a report, change the access code in the *Security Level* field. To restrict access to editing reports, add the REPORT command to the Secured Commands Table.

- Report Description

  This field contains the description that was typed in when the template was created. Type in a new one if you wish.

- Output File Path

  Type in a drive and path specification where the generated report file will reside. For this template, this field permanently overrides the output directory specified in the Reporting Setup window.

  To place the output file in the root directory, enter:
  `C:\`
  where C is the drive where ControlView is installed.

  For all other paths, omit the final backslash.

Important: Whenever you configure a pathname, be sure to start with the drive letter. This is absolutely essential when running ControlView in a multi-drive environment.

- Output File Name

  The generated report file will normally have the same name as the template that was used to generate it. You can type in a different file name, without a file extension.

- File Generation Mode

  Specify either *Create* or *Append*.

  Append means add the data for this reporting session to any data that may already be stored on disk from the previous time the report was generated.

  Create means overwrite the contents of the previous output file with the data for the latest report.

- Include Banner

  When a report is generated, the date and time of report generation and the template name are automatically stored as part of the report file. If you want this information printed as a covering page for the report, specify *Yes*. Otherwise specify *No*.

- Maximum File Size

  If you can estimate the maximum size (in pages) the printed report will likely be, enter the number in the *Maximum File Size* field. Specifying a maximum size ensures that, if a problem generating the report occurs, the computer will not be tied up endlessly.

  A setting of 0, the default setting, means there is no limit on the number of pages that can be printed.

- Page Size

  Setting the length of the page determines how many lines are printed before ControlView sends a form feed command to the printer. Setting the page width determines the number of characters that will be printed on each line.

  The page length ranges from 0 – 256, where 0 means do not generate form feeds. The page width can be as narrow as 40 and as wide as 132 characters. The default is whatever was set as a system-wide default in the Setup option.

- Print on Completion

  When a report is generated, the data is written to a disk file. Specify *Yes* in the *Print on Completion* field if you want the report to be printed when Reporting has finished generating the disk file. Otherwise specify *No*.

- Printer

  This field only applies if you have specified Yes in the *Print on Completion* field.

  The *Printer* field specifies which printer the report should be printed on (if it is to be printed at all). Choices are Printer1, Printer2, Printer3, and Printer4, with the default being whatever was set as a system-wide default in the Setup option.

- Number of Copies

  This field only applies if you have specified Yes in the *Print on Completion* field.

  Specify the number of copies to be printed.

- Delete After Printing

  This field only applies if you have specified Yes in the *Print on Completion* field.

  Specify *Yes* in the *Delete After Printing* field if you want the file deleted as soon as printing is finished. To keep the file on disk, specify *No*.

- Action

  Specify a command that is to be run automatically when the report has been completed. If the report is printed upon completion, the command is not executed until the report is printed. (If the report does not complete, the command is not executed.)

Example: Entering a Command in the Action Field

To list the report after it is generated, enter this command in the Action field:

```
LIST @RPT_OUTPUT
```

@rpt_output is a special keyword that can be used in the Action field to represent the complete path and file name of the generated report. Although `@RPT_OUTPUT` is only 12 characters, the actual file and path name it represents will likely be longer. The total expanded length of the command in the Action line, must not exceed 80 characters.

@rpt_output is only available in the Action field of a report template; it is not recognized anywhere else in ControlView. It can be used with the /M and /C REPORTON command parameters.

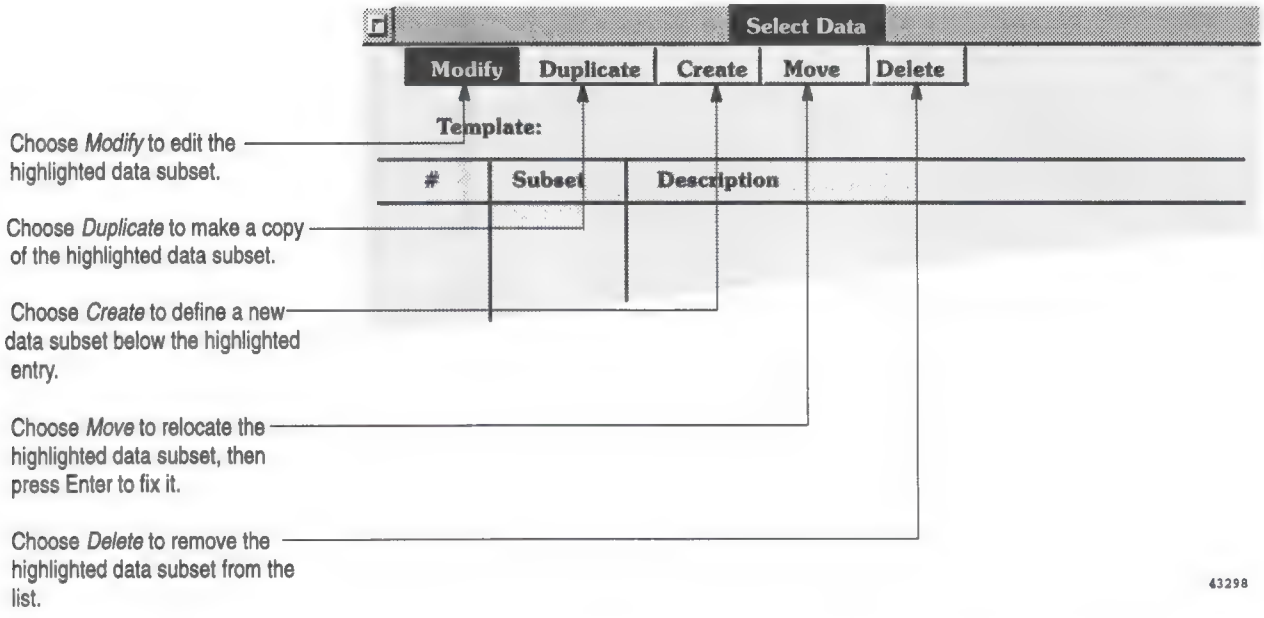
Note that if the report fails to generate - or fails to print if it has been configured for automatic printing - the command named in the Action field will not be executed.

Important: The command named in the Action field may also fail to run if there are too many commands already waiting to execute when the report finishes, because the system is overloaded.

The Data Select Menu

Choose *Data Select* to define the data subsets. The following window appears:

Figure 2.12
The Select Data Window

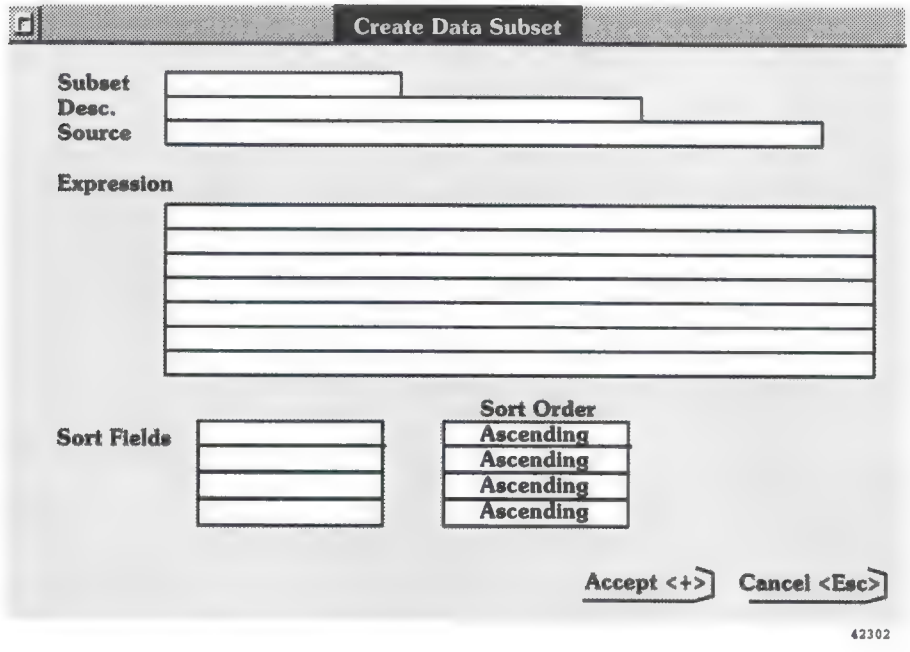


This window allows you to create and manage all the data subsets associated with the report.

Creating a Data Subset

When you choose *Create* from the Select Data window, the following window appears.

Figure 2.13
Create Data Subset



- Subset

  Type in the name (up to twenty characters) for the data subset. This is the name you will specify when defining a table.

- Description

  Type in a description of the data subset. The description is for documentation only.

- Source

  Identify where the data is to be extracted from. Choices are:

  - a log file, including the activity log, the alarm log, or any log files created by Data Logger. You must precede the name of the log file with the @ symbol. There are the possibilities:

    - @activity - the activity log file

    - @alarm - the alarm log file

- *@model* - where *model* is the name of the Data Logger model used to create a Data Logger log file

- *@model*@rpt_generate - where *@rpt_generate* is an optional keyword that can be added to a Data Logger source, to force the generation of the data subset before the report is generated. This is used to set memory variables.

- @rpt_no_source - to specify an @rpt_no_source data subset that is used exclusively to initiate memory variables for generating subsequent data subsets. These memory variables can be initialized to constant values or to tags in the database or to other memory variables.

- the name of a data subset that appears further up in the list (provided that the original data subset was an activity or alarm log, *not* a Data Logger model and *not* a @rpt_no_source subset). By naming a data subset you can create a secondary data subset that eliminates further records from the data subset.

- Expression

  Define an expression, to specify which records will be extracted from the file and added to the data subset. The expressions must have the format:

  ```
  IF condition THEN 1 ELSE 0
  ```

  ControlView will examine the log file record by record. What the IF-THEN-ELSE statement says is "IF the *condition* is true for this record, THEN include the record in the data subset, ELSE skip the record."

  Important: If you want to use the entire log file as the data source, leave the expression field blank.

  Expressions are explained fully in the next chapter.

- Sort Fields

  The alarm log and activity log files can be sorted. By sorting the data subset you can control the order that the items are listed in a table in the report.

  To have a data subset sorted, name the field, or fields that should dictate the sequence. The first field is the primary sort field and subsequent fields are secondary.

  Important: You cannot sort data subsets defined from a Data Logger model.

Example: Sort Fields

You want to define a data subset based on the activity log file, and you want the data

- grouped alphabetically by activity, so you specify the ACT_TYPE field as the first (or primary) sort field
- arranged chronologically within each group, so you specify the ACT_TIME_AND_DATE field as the second (or secondary) sort field

---

The log file formats are given in Appendix B, *File Formats*.

- Sort Order

  Specify an ascending or descending sequence for these. The *Sort Order* fields must remain filled even if you have not specified a sort field.

Chapter 3

Defining an Expression

Defining an Expression

An *expression* is a set of instructions that manipulate data. The term refers to the entire instruction. A segment of an expression is called a *statement.*

Example: Expressions vs Statements

```
(tag1 * tag2) AND (tag3 / 2)
```

is an expression

```
(tag3 / 2)
```

is a statement

Expressions are built from:

- values from the database or log files
- mathematical, relational, logical, and bitwise operators
- statistical, database, Data Logger, time functions and memory variables
- IF-THEN-ELSE program logic

Important: If you are familiar with expressions from other ControlView options, note that there are functions available in Reporting that are not available in the other options.

Throughout this chapter the terms TRUE and FALSE are used. In an expression FALSE means zero and TRUE means any non-zero number.

Expressions are used in two different ways. The first is in field definitions to manipulate data before it is printed in the report. The second is in defining data subsets to determine what data is extracted from log files.

Expressions and Fields

When used to define a field, expressions return either floating point numeric values or text strings. Keep this in mind when you define the field's output format. When the expression returns a numeric value, use the numeric field format character (the # character); when the expression returns a text string, use the text field format character (the A character).

Expressions and Data Subsets

Expressions are also used when defining data subsets to determine which records to include. When used for this purpose, an expression will always have this format:

```
IF statement THEN 1 ELSE 0
```

ControlView interprets this expression to mean "if the *statement* is true for this record, then include the record in the data subset else ignore this record." ControlView checks each record in the log file to build the data subset record by record.

Tag Values

To use a tag's value in an expression, specify the tag name to extract the tag's current value from the database. The tag name may stand alone as the entire expression. Database tag values can be either numeric or alphabetic strings.

Fields

To use the contents of a field in an expression, specify the field name. The field name may be part of a statement or may stand alone as the entire expression. See Appendix B, *File Formats*, for a list of field names.

Naming a field is valid:

- when defining a field inside a table
- when defining a data subset

Constants

Constants are used for comparisons and are used in expressions defining fields or data subsets. They are either numeric or text. Text constants must be surrounded by double quotes.

Examples: Constants

Using the number 123.45 as an example, a numeric constant could have any of the following formats:

- integer (123)
- floating point value (123.45)
- scientific notation (0.12345E3)

Two expressions, used to define data subsets, that use constants:

```
IF ACT_TYPE = "ERROR" THEN 1 ELSE 0

IF DLG_VAL(VIN.OIL) > 100 THEN 1 ELSE 0
```

Database Functions

The database functions extract information from the loaded database. There are two types of database functions: those that return numeric values and those that return strings.

Many of the functions check for a specific TRUE/FALSE condition. Such functions return 1 if the condition is TRUE and 0 if the condition is FALSE. For instance, the COMM_ERR() function returns 1 if the specified tag has a communication error.

Table 3.A
Database Functions That Return a Numeric Value

This function	returns this value	and requires this output format	can be used with this type of tag
COMM_ERR(*tag*)	TRUE if a read or write operation for the specified tag produced a communication failure	#	analog, digital, string
TAG_MAX(*tag*)	the maximum value allowed for the specified tag; must be an analog tag	#(15)	analog
TAG_MIN(*tag*)	the minimum value allowed for the specified tag; must be an analog tag	#(15)	analog
TAG_OFFSET(*tag*)	the offset factor for the specified tag; must be an analog tag	#(15)	analog
TAG_SCALE(*tag*)	the scale factor for the specified tag; must be an analog tag	#(15)	analog
ALM_SUPPRESS(*tag*)	TRUE if the tag's alarms are suppressed	#	analog, digital
ALM_FAULT(*tag*)	TRUE if there has been an alarm fault for the specified tag	#	analog, digital
ALM_SEVERITY(*tag*)	the severity of the alarm—a value between 1 and 8, or 0 if the tag isn't in alarm	#	analog, digital
ALM_LEVEL(*tag*)	the alarm level for the tag: a value between 1 and 8, or 0 if the tag isn't in alarm	#	analog, digital
ALM_ACK(*tag*)	TRUE if the tag's alarm has been acknowledged	#	analog, digital
ALM_IN_ALARM(*tag*)	TRUE if the tag is in alarm	#	analog, digital

Important: The alarm (ALM) functions are only relevant if the Alarming option is installed.

Table 3.B
Database Functions That Return a Text String

This function	returns this text string	and requires this output format	can be used with this type of tag
TAG_DESC(*tag*)	the tag's description	A(32)	analog, digital, string
TAG_TYPE(*tag*)	the type of tag (analog or digital or string)	A(9)	analog, digital, string
TAG_ADDR(*tag*)	the tag's address	A(31)	analog, digital, string
TAG_NODE(*tag*)	the node for the specified tag	A(8)	analog, digital, string
TAG_SCAN(*tag*)	the tag's scan class, a letter between A & H, S1, S2 or S3	A(2)	analog, digital, string
TAG_UNITS(*tag*)	the units of the specified tag	A(6)	analog
TAG_OFFLABEL(*tag*)	the tag's off label, valid only with digital tags	A(10)	digital
TAG_ONLABEL(*tag*)	the tag's on label, valid only with digital tags	A(10)	digital

Examples: Database Functions

```
TAG_DESC(vin.level)
```
Extracts the tag's description from the database

```
TAG_UNITS(vin.level)
```
Extracts the tag's units label from the database

```
ALM_IN_ALARM(vin.level)
```
Returns 1 if the tag is in alarm or 0 if it isn't

```
IF COMM_ERR (vin.level)
          THEN 'Communication error' ELSE ""
```
Returns a message if a communication error was detected while updating the value for the tag vin.level

```
IF ALM_IN_ALARM (vin.level)
          THEN "vin.level is in alarm"
ELSE "vin.level is NOT in alarm"
```
Returns a message indicating whether the tag vin.level is or isn't in alarm.

The IF-THEN-ELSE construction is explained more fully later in this chapter.

Statistical Functions

Functions provide statistical information on the contents of a data subset. The COUNT() function can be used with any data subset; the remaining nine functions make sense only with data subsets based on Data Logger files.

The statistical functions perform running calculations on the entire data subset, so that the count, mean, etc., are updated each time a record is read.

To report only statistical functions derived from a data subset, create a table with only a footer (and no header or body). Include the statistical functions in the footer.

Important: The following functions can only be used to define fields inside a table and cannot be used in a data subset selection expression.

Table 3.C
Statistical Functions

This function	returns this value
COUNT()	the number of records in the data subset
MEAN(*tag*)	the statistical mean for the tag in the data subset
MIN(*tag*)	the smallest value for the tag in the data subset
MAX(*tag*)	the largest value for the tag in the data subset
SD(*tag*)	the standard deviation of the tag in the data subset
TOTAL(*tag*)	the total value of the tag in the data subset
ONCOUNT(*tag*)	the number of records in the data subset that were ON, valid for digital tags only
OFFCOUNT(*tag*)	the number of records in the data subset that were OFF, valid for digital tags only
POSEDGE(*tag*)	the number of positive edges (transitions from OFF to ON) in the data subset, valid for digital tags only
NEGEDGE(*tag*)	the number of negative edges (transitions from ON to OFF) in the data subset, valid for digital tags only

Example: Statistical Functions

`COUNT ( )`
Returns the number of records in the data subset

`MEAN( vin.level )`
Returns the mean value recorded in a Data Logger file for the tag vin.oil.

Time and Date Functions

The following functions manipulate times and dates. These functions use or return a time and date argument (called *tad*), except the REL_TIME() function, which uses the interval parameter.

Important: The time and date parameter and the interval parameter must be surrounded by double quotes.

The Time and Date Parameter

The time and date parameter has two formats: written out or in international date format.

When written out, the time and date parameter has this format:

MMM DD YYYY hh:mm:ss
(abbreviated to MMM DD YY hh:mm:ss)

The international date format is:

YYYY–MM–DD hh:mm:ss
(abbreviated to YY–MM–DD hh:mm:ss)

Example: Time and Date Parameter

The following are valid time and date parameters:

```
"aug 18 1989 17:30:00"
"AUG 18 89 17:30"
"89-08-18 23:59:59"
```

The Interval Parameter

The interval parameter, used by the REL_TIME() function, specifies a whole number followed by a single keyword: years, months, days, hours, minutes, or seconds.

Example: The Interval Parameter

```
NOW() - REL_TIME("150 minutes")
```
Subtracts 2.5 hours from the time the report is generated.

Important: You could not have specified "2.5 hours", since fractions aren't allowed, and you couldn't have specified "2 hours 30 minutes" since only one keyword is allowed in the parameter.

The following functions can be used anywhere in the report, including a data subset.

Table 3.D
Time & Date Functions - Used Anywhere

This function	returns	and requires this output format
NOW()	the time and date when the report is generated	A(20)
TODAY()	midnight on the morning the report is generated	A(20)
GET_TIME("*tad*")	extracts the time from a time & date argument	A(8)
GET_DATE("*tad*")	extracts the date from a time & date argument	A(11)
REL_TIME("*interval*")	used to add or subtract a specific number of years, months, days , hours, minutes, or seconds from a time and date. This function is used only to add or subtract times, never alone.	n/a
ABS_TIME("*tad*")	used to compare a time and date with a specific time and date. This function is used only to compare 2 times, never alone.	n/a

The following functions can only be used inside a table.

Table 3.E
Time & Date Functions - Used Only in Tables

TAD_OF_FIRST_REC()	used to access the time and date when the first record was created. No argument is used with this function.	A(20)
TAD_OF_LAST_REC()	used to access the time and date when the last record was created. No argument is used with this function.	A(20)

Examples: Time and Date Functions

```
NOW ( )
```
Time stamps the report (e.g. 90–02–27 15:30:00).

```
TAD_OF_LAST_REC ( )
```
Returns the time and date for the last record in the table.
(For example, 14:20:00 if the time was 2:20 pm.)

```
GET_TIME( NOW() )
```
Returns the time that the report was generated.
(For example, 17:30:00 if the time was 5:30 pm.)

```
GET_TIME( ACT_TIME_AND_DATE )
```
Returns the time for each record in the activity log file data subset in the format hh:mm:ss.

```
NOW() > ABS_TIME("May 12 1990")
```
Returns 1 if the report is generated after May 12, 1990, 0 otherwise.

```
(NOW() - REL_TIME("2 hours"))
```
Subtracts 2 hours from the time the report is run.

```
IF (ACT_TIME_AND_DATE >
(NOW() - REL_TIME("2 HOURS"))) THEN 1 ELSE 0
```
Defines a data subset with only those records in the activity log file that were recorded in the last two hours.

```
IF ((ACT_TIME_AND_DATE >= TODAY() +
    REL_TIME("2 HOURS")) AND
    (ACT_TIME_AND_DATE <= TODAY() +
    REL_TIME("6 HOURS")))
THEN 1 ELSE 0
```
Returns 1 if the activity log record was recorded between 2:00 a.m. and 6:00 a.m. of the day the report is generated.

The IF-THEN-ELSE construction is explained later in this chapter.

Data Logger Functions

Data Logger files have a complex file format. Each record in the file is a snapshot of all the tags specified in the particular Data Logger model. One record may contain data for one tag or hundreds. Given the complexity of the Data Logger file format, twelve functions have been provided to make it easier to access these files.

These functions can be used when defining a field inside a table or when defining a data subset.

The following functions can be used anywhere in the report, including a data subset.

Table 3.F
Data Logger File Access Functions

This function	returns this value	and uses this output format	can be used with this type of tag
DLG_VAL(*tag*)	the logged value for the specified tag	########.######	analog, digital, string
		A(*n*) for string tags (where *n* is in the range 1–82)	
DLG_COM_ERR(*tag*)	the communication status bit value is 1 if the tag is in error, otherwise 0	#	analog, digital, string
DLG_ON_SCAN(*tag*)	the on scan status bit value is 1 if tag is on scan, otherwise 0	#	analog, digital, string
DLG_ALM_SUPPRESS(*tag*)	the alarm suppress status bit value is 1 if alarms for this tag are suppressed, otherwise 0	#	analog, digital
DLG_ALM_FAULT(*tag*)	the alarm fault status bit value is 1 if alarm is faulted, otherwise 0	#	analog, digital
DLG_ALM_SEVERITY(*tag*)	the alarm severity, a value between 0 and 8	#	analog, digital
DLG_ALM_LEVEL(*tag*)	the alarm level, a value between 0 and 8	#	analog, digital
DLG_ALM_ACK(*tag*)	the alarm acknowledge status bit value is 1 if the alarm has been acknowledged, otherwise 0	#	analog, digital
DLG_TAG_PRESENT(*tag*)	TRUE (i.e., 1) if the specified tag is in the record, FALSE (i.e., 0) otherwise Not all tags defined in the model will appear in every record if conditional expressions have been defined for the tag.	#	analog, digital, string
DLG_TIME_AND_DATE()	the time and date for the particular snapshot	A(20)	
TAD_OF_MIN(*tag*)	the time and date when the minimum value was recorded	A(20)	analog, digital
TAD_OF_MAX(*tag*)	the time and date when the maximum value was recorded	A(20)	analog, digital

Examples: Data Logger Functions

`DLG_VAL( VIN.LEVEL )`
Returns the values in a Data Logger data subset for tag VIN.LEVEL.

`DLG_COM_ERR(VIN.LEVEL)`
Returns 1 if the record in the Data Logger data subset has the communication error bit set on; returns 0 otherwise.

```
TAD_OF_MIN(VIN.LEVEL)
```
Returns the time and date when the value in the Data Logger data subset for tag VIN.LEVEL was at its minimum.

```
IF DLG_COM_ERR(VIN.LEVEL) THEN "vin.level
communication error" ELSE ""
```
Returns the text string indicating a communication error for each record in the Data Logger data subset that has the communication error bit set on; returns a null string otherwise.

```
IF DLG_TAG_PRESENT(VIN.LEVEL) THEN 1 ELSE 0
```
This expression is used to define a data subset where each record will have data about the tag VIN.LEVEL.

The IF-THEN-ELSE construction is explained later in this chapter.

Other Functions

The PAGE_NO() functions will usually be placed in a report's header or footer. The PAGE_NO() function is valid only in expressions defining fields, not in a data subset definitions.

Table 3.G
The Page Number and Square Root Functions

This function	returns this value	and requires this format
SQRT(*tag*)	the square root of a tag (or of a constant)	varies
PAGE_NO()	the report's page number	###

Example: The SQRT() and PAGE_NO() Functions

```
SQRT(vin.level)
```
Returns the square root of the tag's value.

```
SQRT(25)
```
Returns 5.

```
PAGE_NO()
```
Returns the current page number in the report.

Arithmetic Operators

The arithmetic operators can be used in either a field definition or a data subset definition.

Table 3.H
The Arithmetic Operators

Symbol	Operator	Example
+	addition	tag1 + tag2
–	subtraction	tag1 – tag2
*	multiplication	tag1 * tag2
/	division	tag1 / tag2
MOD, %	modulus (remainder)	tag1 % tag2
***	exponent	tag1 ** tag2

The modulus operator is the remainder of one number divided by another. For example, the remainder of 13 divided by 5 is 3; so 13 % 5 = 3.

Examples: Arithmetic Operators

For these examples, tag1 = 5 and tag2 = 7.

```
tag1 + tag2
```
Returns a value of 12.

```
tag1 * tag2
```
Returns a value of 35.

```
tag1 - tag2
```
Returns a value of –2.

```
tag1 / tag2
```
Returns a value of 0.71.

```
tag1 MOD tag2
```
Returns a value of 5.

```
tag1 ** tag2
```
Returns a value of 78125.

Important: Operator precedence is covered later in this chapter.

Relational Operators

Relational operators compare two values to provide a TRUE or FALSE result. If the statement is TRUE, a value of 1 is returned. If FALSE, 0 is returned. There are six relational operators; each has two different symbols.

Relational operators can be used in expressions that define fields or data subsets.

Table 3.I
The Relational Operators

Symbols	Operator	Example	Value type
EQ, =	equal *	tag1 = tag2	string, numeric
NE, <>	not equal *	tag1 <> tag2	string, numeric
LT, <	less than	tag1 < tag2	numeric
GT, >	greater than	tag1 > tag2	numeric
LE, <=	less than or equal to	tag1 <= tag2	numeric
GE, >=	greater than or equal to	tag1 >= tag2	numeric

*** can be used to compare text strings, including string tags, as well as numeric values**

Examples: Relational Operators

For these examples, tag1 = 5 and tag2 = 7.

```
tag1 > tag2
```
is FALSE, so the expression returns a 0.

```
tag1 LE tag2
```
is TRUE, so the expression returns a 1.

```
tag1 = 5
```
is TRUE, so the expression returns a 1.

```
ACT_TYPE = "ERROR"
```
is TRUE if the activity type for the particular record in the activity log file is "ERROR"; if the record in the activity log is of any other activity type, the expression returns a 0.

Logical Operators

Logical operators determine the validity of one or more statements. The operators return a value of 1 if the statement is TRUE, or a value of 0 if FALSE. There are three logical operators: AND, OR, and NOT.

- AND returns a value of 1 if the statements to the right and to the left of the operator are *both* TRUE
- OR returns a value of 1 if *either* the statement to the left or to the right of the operator is TRUE
- NOT reverses the logical value of the statement it operates on

Important: Any statement which evaluates to a non-zero value is regarded as TRUE. For example, the statement tag1 will be FALSE if the value of tag1 is 0 and TRUE if tag1 has any other value.

Logical operators can be used in expressions that define fields or data subsets.

Table 3.J
The Logical Operators

Symbols	Operator	Example
AND, &&	and	(tag1 < tag 2) AND (tag1 = 5)
OR, ||	or	(tag1 = 5) OR (tag1 = 10)
NOT	negation	NOT(tag1 > tag2)

Examples: Logical Operators

For these examples, tag1 = 5 and tag2 = 7.

```
(tag1 < tag 2) AND (tag1 = 5)
```
Returns a 1 since both statements are TRUE.

```
tag1 && tag2
```
Returns a 1 since both tag1 and tag2 are non-zero (TRUE).

```
(tag1 > tag2) OR (tag1 = 5)
```
Returns a value of 1 since *tag1 = 5* is TRUE.

```
NOT(tag1 < tag2)
```
tag1 < tag2 is TRUE and would return a 1 but the NOT operator reverses the logical value so 0 is returned.

Important: The parentheses are essential in the above expressions.

Bitwise Operators

Bitwise operators allow you to examine and manipulate individual bits within a value. These operators can only be applied to integers, not floating point numbers.

Bitwise operators can be used in expressions that define fields or data subsets.

Table 3.K
The Six Bitwise Operators

Symbol	Operator	Example
&	and	tag1 & 07
|	inclusive or	tag2 | tag1
^	exclusive or (XOR)	tag1 ^ 01
>>	right shift	tag1 >> 1
<<	left shift	tag1 << 2
~	complement	~ tag1

The bitwise operators &, |, and ^ compare two integers or tags on a bit by bit basis. The >>, <<, and ~ operators manipulate a single integer or tag.

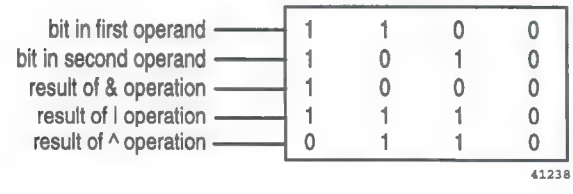
The **bitwise AND (&)** returns an integer with a bit set to 1 if both of the corresponding bits in the original numbers are 1. Otherwise, the resulting bit is 0.

The **bitwise inclusive OR (|)** returns an integer with a bit set to 1 if either or both of the corresponding bits in the original numbers are 1. If both bits are 0, the resulting bit is 0.

The **bitwise exclusive OR (^)** returns an integer with a bit set to 1 if either of the corresponding bits in the original numbers is 1. If both bits are 1 or both are 0, the resulting bit is 0.

The bitwise operators &, |, and ^ are illustrated below:

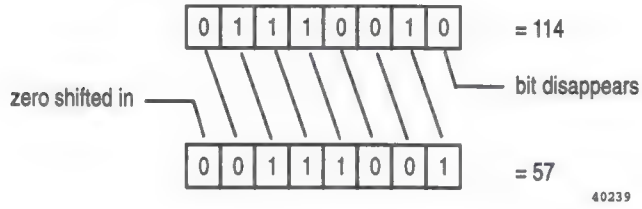
Figure 3.1
The &, |, and ^ Bitwise Operators



The **shift right operator** (>>) shifts the bits within the left operand by the amount specified in the right operand. The bit on the right disappears.

Figure 3.2 shows the result of the expression 114 >> 1 = 57.

Figure 3.2
The Shift Right Operator



Either a 0 or a 1 is shifted in on the left, depending on whether the integer is signed or unsigned. With unsigned integers, 0 is always shifted in on the left; with signed integers, a 0 is shifted in when the number is positive (i.e. the leftmost bit, the sign bit, is 0), and a 1 is shifted in when the number is negative (i.e. the leftmost bit, the sign bit, is 1). In other words, with signed integers, the sign of the number is always maintained.

The **shift left operator** (<<) shifts the bits within the left operand by the amount specified in the right operand. The bit on the left disappears and a 0 is always shifted in on the right.

The **bitwise complement operator** (~) reverses every bit within the number, so that every 1 bit becomes a 0 and vice versa.

Examples: Bitwise Operators

For these examples tag1 = 5 (binary 0101), tag2 = 2 (binary 0010).

```
tag1 & tag2
```
Returns 0 (binary 0000).

```
tag1 | tag2
```
Returns 7 (binary 0111).

```
tag1 ^ tag2
```
Returns 7 (binary 0111).

```
tag1 >> 1
```
Returns 2 (binary 0010).

```
tag1 << 1
```
Returns 10 (binary 1010).

```
~ tag1
```
Returns 10 (binary 1010).

Operator Precedence

Three rules determine the order for evaluating an expression containing more than one operator.

1. When two operators have unequal precedence, the operator with the highest precedence is evaluated first.
2. When two operators have equal precedence, they are evaluated from left to right.
3. To change the normal order of precedence, enclose the statement in parentheses.

Here are the operators from highest to lowest precedence:

Table 3.L
Operator Precedence

Precedence Level	Name	Symbols
1 (highest)	parentheses	( )
2	Logical negation Bitwise complement	NOT ~
3	multiplication division modulus (remainder) exponent logical and bitwise and bitwise shift right bitwise shift left	* / MOD, % ** AND, && & >> <<
4	addition subtraction logical or bitwise inclusive or bitwise exclusive or the statistical, database, Data Logger and Time functions	+ – OR, || | ^
5 (lowest	equal not equal less than greater than less than or equal to greater than or equal to	EQ, = NE, <> LT, < GT, > LE, <= GE, >=

Examples: Operator Precedence

For these examples, tag1 = 5, tag2 = 7 and tag3 =10.

```
(tag1 > tag2) AND (tag1 < tag 3)
```

is evaluated in this sequence:

1. tag1 > tag2 = 0
2. tag 1 < tag3 = 1
3. 0 AND 1 = 0

The expression evaluates to 0.

```
tag1>tag2 AND tag3
```

is evaluated in this sequence:

1. tag2 AND tag3 = 1

2. tag1 > 1 = 1

The expression evaluates to 1.

```
NOT tag1 AND tag2 > tag3 ** 2
```

is evaluated in this sequence:

1. NOT tag1 = 0
2. 0 AND tag2 = 0
3. tag3 ** 2 = 100
4. 0 > 100 = 0

The expression evaluates to 0.

IF-THEN-ELSE

The IF-THEN-ELSE structure allows an expression to return different values depending on the outcome of an expression.

The format of the IF-THEN-ELSE structure is:

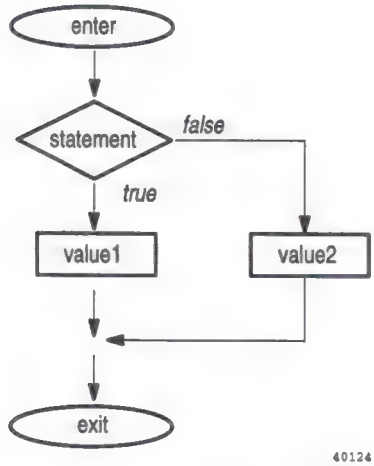
```
IF statement THEN value1 ELSE value2
```

If the *statement* is TRUE then the expression will return *value1*; if the *statement* is FALSE then the expression returns *value2*. Keep in mind that the *statement* is a mathematical equation and that TRUE means a non-zero value, and FALSE means zero.

Important: The ELSE portion of the statement is required.

The IF-THEN-ELSE structure is illustrated below.

Figure 3.3
The IF-THEN-ELSE structure



Important: In an IF-THEN-ELSE statement, all return values must be the same data type, either all strings or all numbers.

In Figure 3.3, value1 and value2 must either both be strings or both be numbers.

Example: Invalid IF-THEN-ELSE statement

This statement mixes data types and is therefore invalid:

```
IF ACT_TYPE = "ERROR" THEN 1 ELSE ""
```

This statement is correct

```
IF ACT_TYPE = "ERROR" THEN 1 ELSE 0
```

Example: Valid IF-THEN-ELSE String Values

If the expression shown in Figure 3.3 is to return a string, here are some possible string values for value1 and value2, :

```
"Within range"
"Out of range"
"Communication error"
""
```

Example: Valid IF-THEN-ELSE Numerical Values

If the expression shown in Figure 3.3 is to return a number, here are some possible numerical values for value1 and value2:

```
0
1
(tag_1  -  tag_2)
(tag_3 / 18)
```

Note that tag_1, tag_2 and tag_3 are analog or digital tags in the database.

Alarm, statistical, database, time and date, Data Logger, page number, square root and Set_Var functions can be used to calculate the return values in IF-THEN-ELSE statements. In particular, the Set_Var function can be used to assign memory variables as the result of an IF-THEN-ELSE statement. Remember that the Set_Var function always returns the value 1.

Example: Using Set_Var in an IF-THEN-ELSE Statement

In the expression :

```
If (tag_1 > 100)
then Set_Var ("tag_1_range","High")
Else Set_Var ("tag_1_range","Normal")
```

The returned value will always be 1, regardless of the value of tag_1.

The Set_Var function can be used to set multiple memory variables in the same IF-THEN-ELSE statement.

Example: Using Multiple Set_Vars Within an IF-THEN-ELSE Statement

The following statement assigns three memory variables. It could be used as a return value in an IF-THEN-ELSE statement.

```
(Set_Var("first_var", 30) +
Set_Var("second_var", tag_17) +
Set_Var ("third_var", "Var # 3"))
```

Each of the three calls to the Set_Var function returns a 1, so the total expression returns the value, 3. This is treated as TRUE in a data subset selection expression, since it is non zero.

If a FALSE return value (the value 0) is required, the expression could be rewritten as:

```
(Set_Var("first_var", 30) +
Set_Var("second_var", tag_17) +
Set_Var ("third_var", "Var # 3") - 3 )
```

Nested IF-THEN-ELSE structure

It is common to nest an entire IF-THEN-ELSE structure inside another IF-THEN-ELSE structure.

When nesting IF-THEN-ELSE structures, it is important to remember that an ELSE is associated with the last IF that isn't already paired with an ELSE.

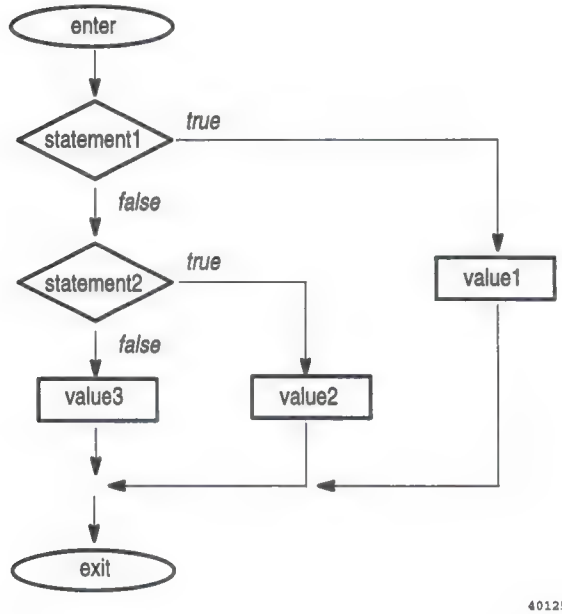
Example 1: Nested IF-THEN-ELSE

This expression:

```
IF (statement1) THEN (value1)
ELSE IF (statement2) THEN (value2)
          ELSE (value3)
```

has this interpretation:

Figure 3.4
Nested IF-THEN-ELSE



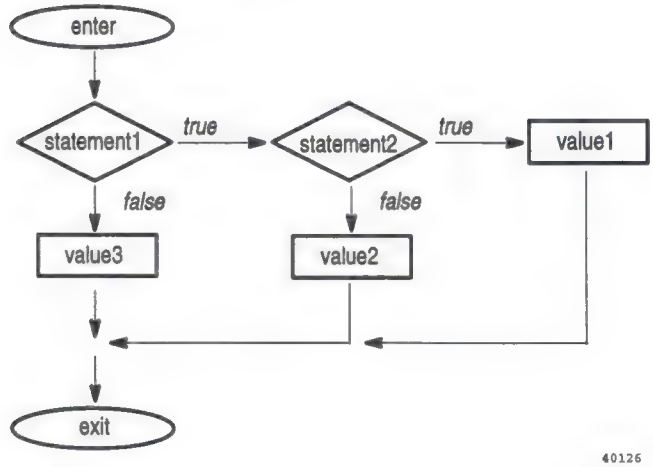
Example 2: Nested IF-THEN-ELSE

This expression:

```
IF (statement1) THEN
          IF (statement2) THEN (value1)
          ELSE (value2)
ELSE (value3)
```

has this interpretation:

Figure 3.5
Nested IF-THEN-ELSE



Comments

Comments begin with an exclamation point; the rest of the line is not evaluated.

Example: Comments in ControlView Expressions

In several lines such as:

```
IF a>b ! temp a greater than temp b
    THEN c ! shut valve
ELSE d ! do nothing
```

only those parts preceding the exclamation points are evaluated.

In a single line such as:

```
IF a>b THEN c ELSE d ! if/then expression
```

only that part preceding the exclamation point is evaluated.

Expression Formatting

Tabs, new line characters, and multiple spaces are ignored in an expression, so a large expression can be condensed to make maximum use of the editor space.

Example: Expression Formatting

An expression can be formatted to make the logic easier to follow, like this:

```
IF (t1 > t2) THEN 0
ELSE IF (t1 > t3) THEN 2
           ELSE 4
```

The same expression can be condensed to the following:

```
IF (t1>t2) THEN 0 ELSE IF (t1>t3) THEN 2 ELSE 4
```

Important: Do not allow tag names, keywords, function names or function arguments—with the exception of text strings—to span more than one line. A line of text can be up to 132 characters long.

Memory Variables Using SET_VAR

The set_var function is an advanced feature enabling you to define memory variables.

The set_var function can be called in any expression, whether in a data subset definition or field definition. The function takes two arguments: the first is a string (within quotes) naming the memory variable, the second is the value for the variable.

A valid set_var function has *two* arguments:

- First argument names the variable. It is a string, enclosed in quotes. It consists of alphabetical, numeric and underscore characters only. It cannot include leading or imbedded blanks. The maximum length is 20 characters.
- Second argument is the value of the variable. The maximum length is 132 characters.

The set_var function always returns a value of 1.0 (unless there has been a runtime error).

If the memory variable specified in the set_var function doesn't exist, it is created. If it does exist, the old value is replaced with the new one. The value can be numbers, ascii strings or Time and Dates.

In keeping with the rest of the system, the set_var function can handle floating point values in the range of 3.4×10^{-38} to $3.4 \times 10^{+38}$, with 7 digits of precision.

Examples: The set_var Function

The following are all *valid* calls to the set_var function:

```
SET_VAR("a_num", 3482)
SET_VAR("a_sum", (3 + 4 + 5))
SET_VAR("a_tad", ACT_TIME_AND_DATE)
SET_VAR("dlg_tad", DLG_TIME_AND_DATE())
SET_VAR("a_string", "Hello young lovers")
SET_VAR("a_logical", (tagname >=300))
```

The following are all *invalid* calls to the set_var function:

```
SET_VAR(a_var, 33)              identifier is not in quotes
SET_VAR("a_var")                too few arguments
SET_VAR("a_var", 3, 0)          too many arguments
SET_VAR(" leading_blank", 0)    invalid identifier
SET_VAR("imbedded blank", 0)    invalid identifier
SET_VAR("","")                  invalid identifier
```

A memory variable can be referenced anywhere a tag name or data subset field name can be. Inside a table, Reporting will check any identifier, first to see if it matches a field in the data subset, then to see if it is a tag in the database, and finally, to see if it is a memory variable. Outside a table, Reporting checks the database then the memory variables.

Example: Defining a Memory Variable

If a Data Logger data subset is defined to contain snapshots that include a specific tag, the last value for that tag can be stored using this expression:

```
IF (DLG_TAG_PRESENT(tagname))
   THEN Set_Var( "last_tag_value", DLG_VAL( tagname ))
   ELSE 0
```

You could then reference this variable with the expression:

```
IF ((last_tag_value) > 100)
   THEN "tagname larger than 100"
   ELSE "tagname less than 100"
```

and specify a format "A(25)".

One use of memory variables is to allow information from a data subset to be included in a report, but outside a table.

Example: Specifying Data Subset Information Outside a Table

In a data subset derived from the Activity Log, use the selection expression:

```
If (Act_Type == "Error")
      Then Set_Var("TAD_last_error", ACT_TIME_AND_DATE)
Else 0
```

Then the time and date of the last error listed in the Activity Log can be used anywhere in a report, including page headers and footers.

Another use of memory variables is to dynamically select data from a source based on the contents of another data subset.

Example: Selecting Records from a Data Log File that are Determined by the Alarm Log

This example selects records from a data log file for the period that the most recent tag was "In Alarm", which is determined from the Alarm Log.

In a data subset derived from the Alarm Log, select all the "In Alarm" and "Out of Alarm" records by using the selection expression:

```
If (ALM_TYPE == "InAlm")
      Then (Set_Var("In-Alarm_TAD", ALM_TIME_AND_DATE)
           + Set_Var("Out_Alarm_TAD", Now() ) )
Else If(ALM_TYPE == "OutAl")
      Then (Set_Var("Out-Alarm_TAD", ALM_TIME_AND_DATE)
Else 0
```

Note that each time an "In Alarm" record is encountered, the corresponding "Out of Alarm" time and date variable is reset. This handles the situation where the last "Out of Alarm" record occurs before the last "In Alarm" record. This expression will reflect the fact that the tag is still "In Alarm". If the "Out of Alarm" time and date were not reset, this next expression would generate an empty data subset.

In a subsequent data subset, derived from a data log model that contains data for the alarmed tags, use the selection expression:

```
If (DLG_TIME_AND_DATE() >= In_Alarm_TAD ) AND
   (DLG_TIME_AND_DATE() <= Out_Alarm_TAD )
    Then 1 Else 0
```

This expression will select all the data log records for the period that the most recent tag was in alarm.

Notes on Set_Var and Memory Variables

Set_Var is an advanced feature. Before using the Set_Var function, read the following:

- Set_Var does not return the value of the memory variable; it always returns 1.0 (unless there is a runtime error.)
- When defining a Memory Variable, the identifier is enclosed in quotes as the first argument; when retrieving the value, the identifier does not appear in quotes but is treated like a tag name or data subset field name.

- The expression used to select data for a data subset has the form:

  `if` *statement* `then 1 else 0`

  The entire *statement* is evaluated before Reporting determines whether the record is to be included within the data subset. This means that if you use the Set_Var function in the *statement*, the function will be called, and the variable set, for every record in the file, regardless of whether the statement evaluates to true or not.

Example: Selecting Data

```
IF (alm_type = "INALM") and
      set_var ("text", "alarm occurred")
then 1 else 0
```

"text" would have the value "alarm occurred" even if alm_type was not INALM.

If you want to set the memory variable only for those records that are INALM, the expression should have this format:

```
if (alm_type = "INALM")
then set_var("text", "alm occurred")
else 0
```

- Memory Variables cannot be referenced until they are defined. This is important when they are used within both the report and a data subset definition. Data subsets—with the exception of those derived from Data Logger files—are generated before the first line of the report itself is generated. Because of the potential size of Data Logger files, data from these files is read only when it is needed inside a table, ***not*** before the report is generated. This information is summarized in Table 3.M.

The @rpt_generate and @rpt_no_source Keywords

Use these keywords when working with memory variables.

@rpt_generate

When a report is generated, all data subsets with the exception of Data Logger data subsets, are generated before the report is printed. Data Logger data subsets, however, are not generated until they are referenced as the report is being printed. You can force the system to generate a Data Logger subset before the first line of the report is printed. Simply add the keyword *@rpt_generate* to the source field of the data subset.

Using this keyword increases the time required to generate the report. Only use rpt_generate if the set_var function is being used in the selection expression of a Data Logger subset.

Example: Adding the @rpt_generate keyword

A Data Logger data subset with the source *@model*
is not generated until it is referenced in a table. Any memory variables defined in the data subset selection expression cannot be used until after the table is printed.

A Data Logger data subset with the source *@model@rpt_generate*
is generated along with the other data subsets, before the report is printed. Thus any memory variables defined in the selection expression can be used in subsequent data subset expressions, page headers or footers, or elsewhere in the report.

@rpt_no_source

Data subsets can be created that specify the keyword *@rpt_no_source* as the source. These @rpt_no_source data subsets are not true subsets but exist only to have their selection expressions evaluated once and only once. Use them to initialize memory variables that are to be used while generating subsequent data subsets. Using this keyword has little effect on memory utilization.

- any number of data subsets can be specified with the source *@rpt_no_source*
- @rpt_no_source data subsets can appear anywhere in the list of subsets
- @rpt_no_source data subsets cannot be used by a table or be referenced by any other data subset

Table 3.M
Defining and Using Memory Variables

If Memory Variables are first defined:	It will be recognized here:	It will NOT be recognized here:
• in an expression defining a "normal" data subsets (excluding Data Logger files) • @rpt_generate Data Logger files • @rpt_no_source data subsets	• in any "normal" data subset definition appearing later in the list of data subsets • in any expression within the report itself • in any expression used to define a data subset derived from a Data Logger file	• in any "normal" data subset definition appearing earlier in the list of data subsets
• in an expression within the report	• in any expression later in the report • in an expression used to define a data subset derived from a Data Logger file, if the associated table appears ***later*** in the report	• in any expression earlier in the report • in any "normal" data subset definition • in an expression used to define a data subset derived from a Data Logger file, if the associated table appears ***earlier*** in the report
• in an expression defining a data subset derived from Data Logger files, without @rpt_generate keyword	• within the table that uses the data subset • in the report, so long it appears after the table where the data subset is first accessed	• in any "normal" data subset definition • in the report before the table where the data subset is first accessed

Parameters

Many of the fields defined in a report template will contain the name of a database tag or a data subset field. When the report is generated, the value from the database or data subset is used in the expression.

There may be times, however, when you don't want to name the tag or field until the report is actually generated. For this reason, Reporting lets you specify a parameter in an expression instead of a tag or field.

Example: Using Parameters

You are defining reports for a plant that cans peas and beets. The machinery used in both processes is identical, but the vegetables are different. To report on these processes, you could create two templates, one using pea-related tags and one using beet-related tags; in every other respect, the templates would be identical. A simpler method is to create one template that substitutes parameters for the tag names. The operator must then specify what tags Reporting will use when the report is generated.

When the generic template is created, a parameter is used whenever an expression expects a tag or field name. The parameter consists of the pound sign (#) followed by a number between 1 and 50, like this: #3.

When you generate the report, you must specify which tags or fields will replace the parameters. There are two ways of identifying the tags and fields to substitute: with a parameter file, or directly from the command line.

The Parameter File

When using a parameter file, you must create one entry for each unique parameter in the template. If a template contains three parameters, (#1, #7, and #23, for example), then the parameter file must explicitly name three tags or fields. If the template contains a parameter with a number that is not in the parameter file, an error message will appear.

The parameter file will contain the parameter, an equal sign, and the specific tag or field name, like this:

```
#1 = peas.off

#7 = peas.on

#23 = peas.weight
```

Use a DOS text editor to create the parameter file; name the file *without* a file extension and save the file in the \ACCESS\PAR directory.

Generate the report with the REPORTON command, using the /P parameter to name the parameter file. For example, if the template is called CANNING, and the parameter file is called PEAS, you would generate the report using the parameter file with this command:

```
REPORTON canning /Ppeas
```

The report would be generated using the tags or fields specified in the file.

It is possible for one template to use more than one parameter file (one at a time), and for one parameter file to apply to more than one template. A single parameter file might contain 50 parameters. The system might have ten generic templates, identical except for the tag names. Each template would access the same parameter file, but would use different parameter numbers from the file—the first using #1 through #5, the second #6 through #10, the third using #9, #14, #28, #46, and #49, and so on. Alternatively, the same template might be loaded using parameters from a different file, which would effectively make it a different template.

Substituting Tags from the Command Line

You can also name on the command line the tags or fields that replace parameters. To name tags and fields directly from the command line, use the /T parameter. List each tag or field, and separate them with commas.

For example, if the CANNING template had parameters #1, #2, and #3, and you wanted to use the peas tags, the template command would be:

```
REPORTON canning /Tpeas.off,peas.on,peas.weight
```

The first tag on the command line replaces parameter #1, the second replaces parameter #2, and so on.

If the parameters used in the template aren't numbered consecutively, you must add a "null" or empty parameter to represent the parameters that don't exist. For example, if the CANNING template had two parameters, #1 and #3, and you want the template to use the peas.off and peas.on tags, to generate this report you'd type:

```
REPORTON canning /Tpeas.off,,peas.on
```

The double comma means there is no tag for parameter #2.

Passing Time and Date Arguments at Runtime

Use REPORTON's /T parameter to pass the arguments for the ABS_TIME and REL_TIME functions at runtime. This information can be specified at the command line or from within a parameter file. With the /T parameter, you can define a report to accept these time and date parameters.

Example: Using the Time and Date in a Report

To generate a report on demand and vary the start and end times (such as for a specific batch of a product) use the /T parameter. Replace each time, date or interval parameter with a place holder and use the /T parameter to specify these values at runtime.

```
if ( (DLG_TIME_AND_DATE() >= ABS_TIME( "#1 #2" ) ) &&
    (DLG_TIME_AND_DATE() <= ABS_TIME( "#1 #3" ) ) ) then 1 else 0
```

The resulting command would be:

```
REPORTON shift1 /T92-04-13,08:00:00,16:00:00
```

The report file would be named "shift1", and would contain logged data for April 13 1992, from 8 am to 4 pm.

You could also use the "TODAY()" function and REL_TIME, like this:

```
if ((DLG_TIME_AND_DATE() >= TODAY() + REL_TIME( "#1 #3" )) &&
   (DLG_TIME_AND_DATE() <= TODAY() + REL_TIME( "#2 #3" ))) then 1 else 0
```

and the resulting command would be:

```
REPORTON shift1 /T8,16,hours
```

The file name would be "shift1" and would contain logged data for the current day, from 8 am to 4 pm.

Place Holders in the Action Field

The Action field in the Template Configuration window allows you to specify a command to be run when the report has been completed. You access the Template Configuration window from the Report Editor(Figure 2.3). From the Report Editor menu, choose *Configuration* and the Template Configuration window will open.

The Action field accepts place holders for commands or macros. Place holders (like #1) are only 2 characters, but the value they represent will almost always be longer. The total expanded command length cannot exceed 80 characters.

Note that if the report fails to generate—or fails to print if it has been configured for automatic printing—the command named in the Action field will not be executed. The command named in the Action field may also fail to run if there are too many commands already waiting to execute when the report finishes.

Example: Using Place Holders in the Action Field

When you define the report named *RPT*, you enter these place holders in the Action field:

```
#1 #2 #3
```

Then to generate the report named *RPT*, you enter the following command.

```
REPORTON RPT /TSET,tag1,1
```

In this command, **SET** replaces **#1**, **tag1** replaces **#2**, **1** replaces **#3**. This is the expanded command that must not exceed 80 characters. So when the report is complete, the Action field provides the following command:

```
SET tag1 1
```

Parameters and Text

There is one last use for parameters and that is to insert text into a report at run time. The most obvious example for this use would be to have the operators enter their name when they generate a report.

Within an expression, text strings must appear inside double quotes. If a parameter is to be used to represent a text string, it too must be between double quotes in the report template.

Example: Parameters and Text

To create a field that would display the operator's name when the report is generated, you would use the following expression:

```
"#1"
```

Assume the template is called CANNING. For an operator named John to generate this report and have his name show up in the report, he would use this command:

```
REPORTON CANNING /TJOHN
```

The same parameter can be used both as a string and as a value. For example, if you loaded a template with this command:

```
REPORTON CANNING /TPEAS.OFF
```

then this expression
#1
would return the current value for the tag peas.off

and this expression
"#1"
would return the text string "PEAS.OFF"

Sample Expressions

This section provides a number of sample expressions so that you can see how all the pieces fit together.

Example: Expressions Used To Define Fields

To create a field that shows the alarm status of the tag vin.level, define a field with this output format of `A(12)` and use this expression:

```
IF ALM_IN_ALARM(vin.level)
   THEN "in alarm"
ELSE "out of alarm"
```

Instead of hard-coding the tag name into the expression, you could define the same expression to use a parameter (and supply the tag name when the report is generated), like this:

```
IF ALM_IN_ALARM(#1)
   THEN "in alarm"
ELSE "out of alarm"
```

To display the value of a tag only when the tag is communicating with the PLC, you'd use this expression:

```
IF NOT COMM_ERR(vin.level)
   THEN vin.level
ELSE 0
```

If you had a table with the activity log as data subset, you could use an expression to translate the codes into expanded form, like this:

```
IF ACT_TYPE = "COMM"
           THEN "Communication Status"
ELSE IF ACT_TYPE = "CMD"
           THEN "Operator Command"
ELSE       "System Error"
```

This expression will work with the default activity log setup where these three activities are the only ones that are logged.

`GET_DATE( DLG_TIME_AND_DATE() )`
Returns the date for each snap shot in the Data Logger data subset.

Example: Expressions Used To Define Data Subsets

To generate a data subset that contains all the comments operators entered into the activity log file, use the following expression:
`IF ACT_TYPE = "REM" THEN 1 ELSE 0`

To create a data subset with all the alarms recorded for vin.level, use the following expression:
`IF ALM_TAG_NAME = "VIN.LEVEL" THEN 1 ELSE 0`

To create a data subset from a Data Logger file that has all the records containing values for vin.level, use the following expression:
`IF DLG_TAG_PRESENT(VIN.LEVEL) THEN 1 ELSE 0`

Chapter 4

Generating Reports

In order to generate a report, Reporting must:

- produce the data subsets, if any were defined
- take a snap-shot of the loaded database
- go through the template and produce the report, line by line

If there are any errors in the report template, such as references to invalid field or tag names, they are only discovered when the report is generated, not when the report template was created with the Report Editor.

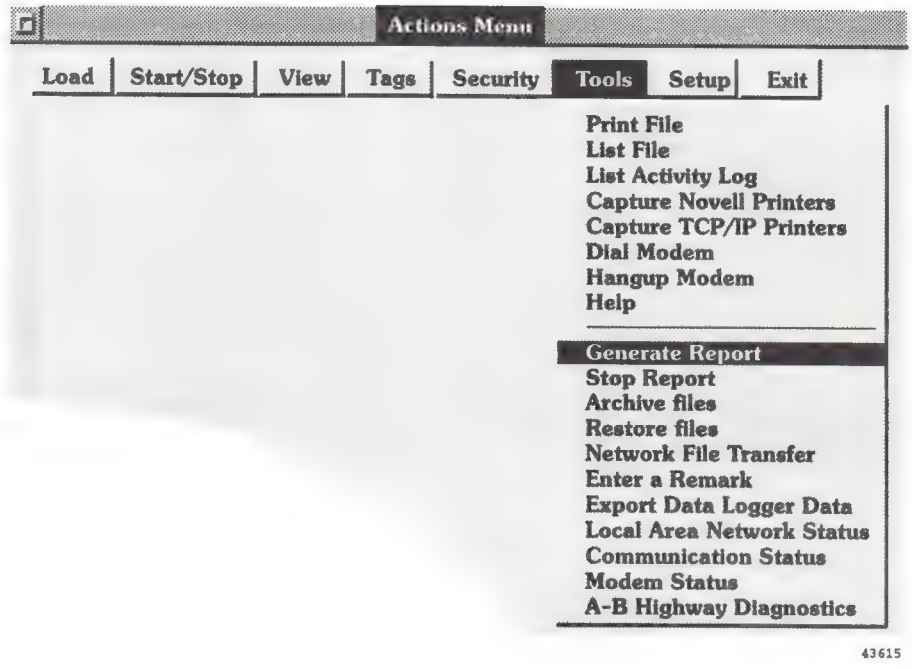
If you are running ControlView on a networked system, consider dedicating one workstation to generating reports, since generating a report will slow down performance of the computer.

Generate Report

To generate a report

1. Choose *Generate Report* under Tools in the Actions Menu.
2. Specify which report you want generated. You can generate more than one report, with each new report being added to a queue until it gets a chance to run.

Figure 4.1
Generating a Report



Stop Report

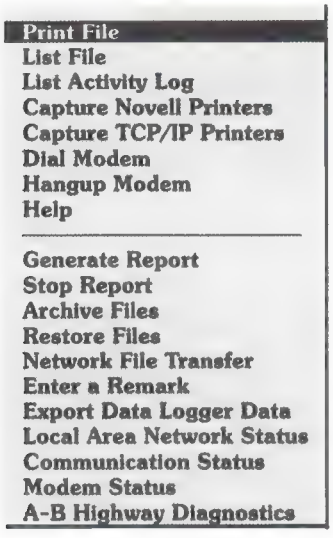
You can let the report run until completion, which may take considerable time if the report uses many data subsets and performs many calculations. Or you can cancel the report before it is completed by choosing *Stop Report* under Tools in the Actions Menu. Specify which report you want to stop generating.

- If the specified report is being generated, processing on that report stops, and the next report in the queue (if any) will begin generation.
- If the specified report is in the queue, it is removed from the queue.

You can have even more control over the reporting queue by using the REPORTON and REPORTOFF commands from the command line. See Appendix A, *Reporting Commands*, for details.

Viewing a Report

All reports are generated as disk files. Depending on how you've configured the report, ControlView can print the report and then delete the file after printing is completed.



If you haven't configured the report to print and delete the file, use the LIST or PRINT commands, or choose *List File* or *Print File* under tools in the Actions Menu, to view the completed report. See Chapter 3, *The Actions Menu*, or Appendix A, *ControlView Commands* in the *ControlView Core User Manual*. The default location for completed reports is the C:\ACCESS\RPT\OUT directory, where C is the drive where ControlView is installed. Unless you've specified a different name for the output file, it will have the same name as the template, without a file extension.

Important: Whenever you configure a pathname, be sure to start with the drive letter. This is absolutely essential when running ControlView in a multi-drive environment.

Example: Viewing a Report

To view a report called SAMPLE, type this command at the command line:

`LIST C:\ACCESS\RPT\OUT\SAMPLE` *press Enter*

Creating Multiple Versions of a Report

Two parameters (/M and /C) in the REPORTON command, enable you to save multiple versions of a report.

/M adds a numerical file extension to the output file name. The first time the report is generated, the report file is assigned the extension *.000*. The second time the report file is assigned the extension *.001* and so on up to *.999*.

If the /M parameter is used when all file extensions 000-999 are already present, an error will be presented and the report will not be generated.

Important: The deletion of reports generated with the /M parameter must be carefully controlled because Reporting assigns the lowest possible number to a new report. If reports .000 to .087 exist and then reports .009, .017 and .027 are deleted, the next four reports that are generated will be assigned the numbers .009, .017, .027, .088; in that sequence.

/C produces an output file whose name is based on the date and time. The file name format is *YYMMDDHH.MMS* (two digits each for year, month, day, hour, and minutes, and one digit for tens of seconds).

The /C and /M parameters can be used with other parameters, but should *not* be used at the same time. However, if both parameters are used together, only the /C is executed, the /M is ignored.

Example: Creating Multiple Versions of a Report's Output File

Run the following command twice:

```
REPORTON salad /M
```

The first time, the file *salad.000* is produced; the second time, the file *salad.001* is produced.

Run the following command on May 12, 1992 at precisely 4:23 pm plus 40 seconds:

```
REPORTON salad /C
```

The file named *92051216.234* is produced.

Report Completed Message

A Report Completed message is displayed whenever a report has been generated. The message shows the name of the output file for the report. This message can be suppressed by using the /S parameter in the command.

Errors

Reporting indicates two types of errors: compile errors and runtime errors.

- Compile errors are displayed on the screen. They are identified as you create the report template.
- Runtime errors are printed on the report and may also be displayed on the screen. They are identified as the report is being generated and include invalid expressions, being out of memory or out of disk space.

Appendix A

Reporting Commands

REPORT

REPORT [*template*]

Calls up the Report Editor.

[*template*] is the name of the report template you wish to edit. If no template is specified, Reporting starts at the Report Template Browser.

REPORTOFF

REPORTOFF [*template*] [/A] [/Q]

Cancels the generation of a report.

If no parameters are specified, this command calls up the Reporting Queue (Figure A.1), a screen that lists all the reports queued to be generated and identifies the report currently being generated.

[*template*] cancels the named report if it is being generated, or removes the report from the queue if it isn't yet being generated.

[/A] cancels the report being generated and all reports in the queue.

[/Q] cancels all reports in the queue while permitting the report that is being generated to complete.

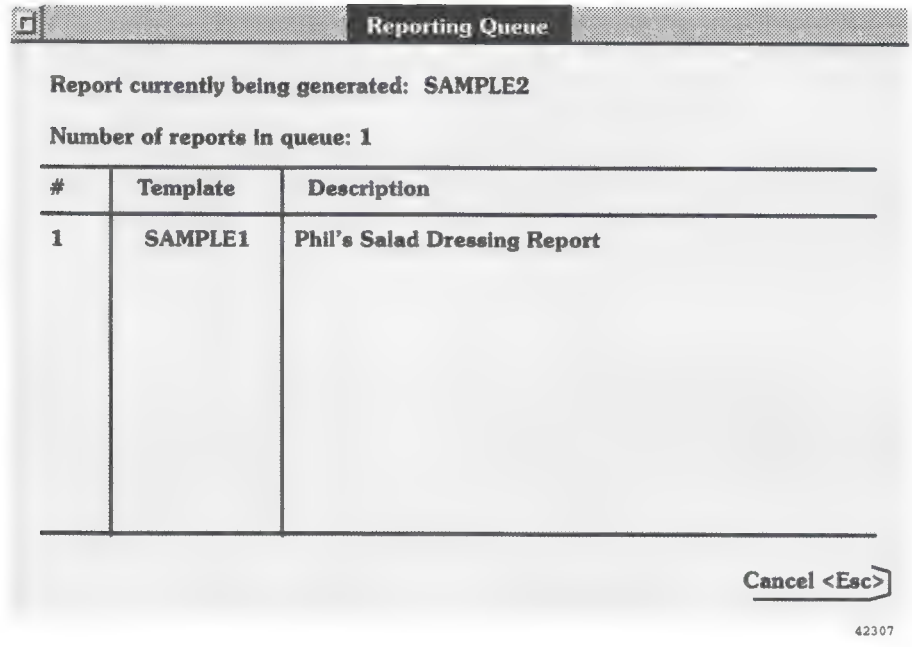
REPORTON

REPORTON [*template*] [/O*out*] [/A] [/*Pfile*] [/T*parm*] [/*n*] [/B] [/N*n*] [/D] [/M] [/C] [/S]

Generates a report.

If no parameters are specified, this command produces a screen that lists all the reports queued to be generated and identifies the report currently being generated.

Figure A.1
Reporting Queue



[*template*] the name of the report you want to generate. If another report is being generated when you issue this command, the requested report will be entered in a queue of reports waiting to be generated.

[/O*out*] the name of the output file created for this report. The /O parameter overrides the output file named in the template setup. You can name a file on the local hard disk, on a floppy disk, or on a network drive.

[/A] the generated report should be appended to the output file. Can be used with the /O parameter.

[/P*file*] where *file* names the parameter file to be used with the report. The parameter file can be used either in the selection criteria when defining data subsets or in expressions when defining fields.

[/T*parm*] where *parm* is a list of command-line parameters, separated by commas. These parameters will be substituted into the template or Action field.

[/*n*] specifies which printer the report is to be printed on, with /1 representing Printer1, /2 representing Printer2 and so on. Even if the report is printed, it will still be saved as a disk file.

[/B] adds a banner page to the front of the report.

[/N*n*]	prints multiple copies of the report, where *n* represents the number of copies. Specifying 0 copies prevents the report from being printed.
[/D]	deletes the disk file after the report has been printed.
[/M]	saves multiple output versions by adding a 3 digit numeric extension to the output filename.
[/C]	saves multiple output versions by using a chronological ouput filename of the form *YYMMDDHH.MMS*.
[/S]	suppresses the Report Completion message.

Examples: The REPORTON Command

`REPORTON`

with no parameters, calls up the Reporting Queue screen.

`REPORTON SHIFT1`

generates the report called shift1. If another report is being generated, shift1 is added to the queue of reports waiting to be generated.

`REPORTON shift1 /Ox:\rpt\shift1`

generates the report called shift1 and stores the results on the network. The completed file can be found in a file called SHIFT1 on network drive X in the \RPT directory. **Note:** this \RPT directory must already exist.

`REPORTON shift1 /3 /B /N2 /D`

prints two copies of the report called shift1 on Printer3 and attaches a banner page to both copies. The file is deleted after the second copy of the report has finished printing.

Appendix B

File Formats

Activity Log

Activity log records have three fields, as shown in Table B.A. The output format is affected by the Activity Log configuration which you define by choosing *Configure Activity Log* under Configure in the Setup menu.

Table B.A
The Activity Log Record Format

This field name	contains	and requires this output format
ACT_TIME_AND_DATE	the time and date in the format YY–MM–DD hh:mm:ss	A(18)
ACT_TYPE	the name of the activity type.[1] Possibilities are: ERROR, error messages CMD, execution of commands APPLC, application activity REM, operator's remarks COMM, communication status WRITE, tag writes READ, tag reads CUST1, CUST2, CUST3, CUST4, C-Toolkit activity	AAAAAA
ACT_COMMENT	the message associated with the activity	A(55) for the 80 column format or A(107) for the 132 column format

[1]These are the default names for the activity type. As explained in the section Configure Activity Log in Chapter 2, *The Setup Menu* of the *ControlView Core User Manual*, it is possible to change these names. If the names have been changed on your system, use the revised names.

Alarm Log

Alarm log records have seven fields, as shown in Table B.B.

Important: When configuring the Alarming application module, it is possible to re-define the messages stored with an alarm record. However, Reporting will not be able to use the Alarm Log file if the default file messages have been changed.

Table B.B
The Alarm Log Record Format

This field name	contains	and requires this output format
ALM_TIME_AND_DATE	the time and date in the format YY–MM–DD hh:mm:ss	A(18)
ALM_TYPE	names the alarm type possibilities are: OUTAL, — out of alarm INALM, — in alarm ACKED, — acknowledged INFLT, — in fault OUTFL – out of fault	AAAAA
ALM_TAG_NAME	the name of the tag in alarm	A(21)
ALM_LABEL	the out–of–alarm label valid only for OUTAL record type	A(10)
ALM_THRESHOLD	the alarm threshold valid only for INALM record type	#(16)
ALM_UNITS	the tag's units, as recorded in the database valid only for INALM record type	A(11)

Important: If you sort the alarm log on a field that doesn't appear in every record, Reporting will generate a runtime error when you try to run the report. For example, if you tried to sort the entire alarm log on the ALM_THRESHOLD field, you'd get an error since this field appears only in the INALM record type. Had you first extracted all INALM records, you could then sort on the ALM_THRESHOLD field.

Data Logger Files

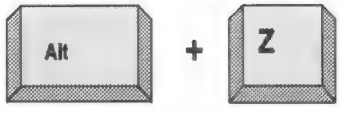
The file format for Data Logger files is much more complex than for Activity Log and Alarm Log files. For this reason, you must use the Data Logger functions, presented in Chapter 3, *Defining an Expression*, to extract data from a Data Logger file.

Appendix C

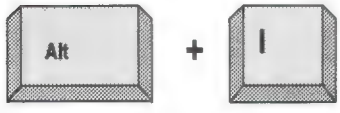
Keys

Menu Shortcuts

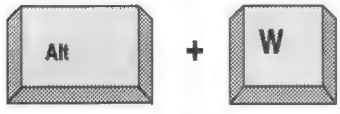
Edit



switch back and forth between menu and edit window

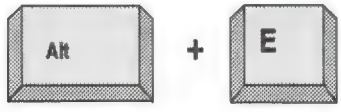


insert contents of buffer into template



write line or field to buffer (duplicate)

delete character, field, summary or page break icon. Only the field is saved in the buffer

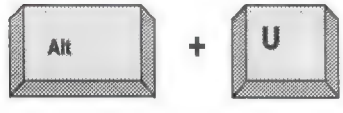


edit object

Objects

create field

create table

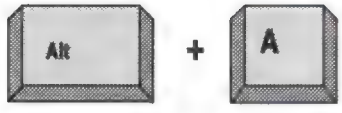


create summary

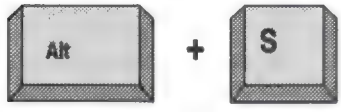
Format

draw mode

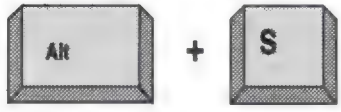
Search



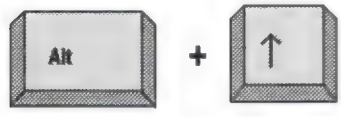
search again

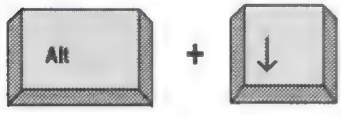


search



replace





direction of search/replace

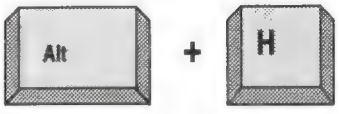
File

save and exit

exit without saving

quick save, and continue editing

Help



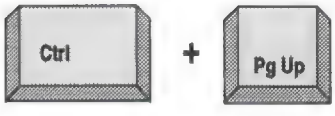
display help screen

Editing Keys

beginning of line

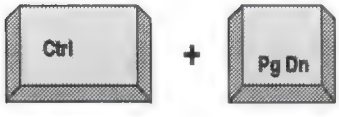
end of line

up one page



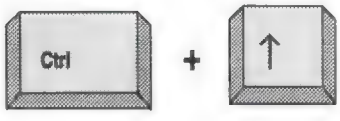
top of document

down one page



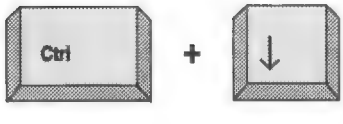
bottom of document

up one line



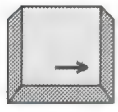
top of window

down one line

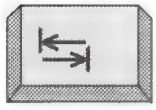


bottom of window

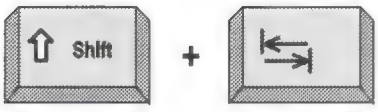
left one character



right one character

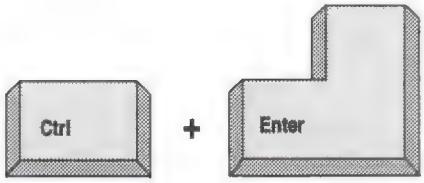


right eight characters

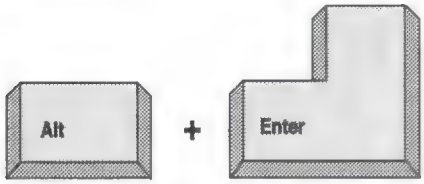


left eight characters

toggle between insert and overwrite



insert a new line above the cursor



insert a new line below the cursor

delete from cursor to end of line

delete entire line that cursor is on. The deleted line is copied to the buffer—overwriting the buffer's contents—and can be inserted elsewhere in the template

Symbols

A

B

C

V

W

ALLEN-BRADLEY
A ROCKWELL INTERNATIONAL COMPANY

As a subsidiary of Rockwell International, one of the world's largest technology companies — Allen-Bradley meets today's challenges of industrial automation with over 85 years of practical plant-floor experience. More than 11,000 employees throughout the world design, manufacture and apply a wide range of control and automation products and supporting services to help our customers continuously improve quality, productivity and time to market. These products and services not only control individual machines but integrate the manufacturing process, while providing access to vital plant floor data that can be used to support decision-making throughout the enterprise.

With offices in major cities worldwide

WORLD HEADQUARTERS
Allen-Bradley
1201 South Second Street
Milwaukee, WI 53204 USA
Tel: (1) 414 382-2000
Telex: 43 11 016
FAX: (1) 414 382-4444

EUROPE/MIDDLE EAST/AFRICA HEADQUARTERS
Allen-Bradley Europe B.V.
Amsterdamseweg 15
1422 AC Uithoorn
The Netherlands
Tel: (31) 2975/43500
Telex: (844) 18042
FAX: (31) 2975/60222

ASIA/PACIFIC HEADQUARTERS
Allen-Bradley (Hong Kong) Limited
Room 1006, Block B, Sea View Estate
28 Watson Road
Hong Kong
Tel: (852) 887-4788
Telex: (780) 64347
FAX: (852) 510-9436

CANADA HEADQUARTERS
Allen-Bradley Canada Limited
135 Dundas Street
Cambridge, Ontario N1R 5X1
Canada
Tel: (1) 519 623-1810
FAX: (1) 519 623-8930

LATIN AMERICA HEADQUARTERS
Allen-Bradley
1201 South Second Street
Milwaukee, WI 53204 USA
Tel: (1) 414 382-2000
Telex: 43 11 016
FAX: (1) 414 382-2400

Publication 6190-6.5.8 November 1992
Supersedes Publication 6190-6.5.8 November 1990

PN 504629602